QUEUEING MODELS FOR FILE MEMORY OPERATION

by

PETER JAMES DENNING

B.E.E., Manhattan College
(1964)

SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Signature of Author Peter J. Denning
Department of Electrical Engineering, May 21, 1965

Certified by Jack B. Dennis
J.B. Dennis, Assoc. Prof. Elec. Eng.

Accepted by Truman S. Gray
Chairman, Departmental Committee on Graduate Students

QUEUEING MODELS FOR FILE MEMORY OPERATION

by

PETER JAMES DENNING

Submitted to the Department of Electrical Engineering on May 21, 1965, in partial fulfillment of the requirements for the degree of Master of Science.

ABSTRACT

A model for the auxiliary memory function of a segmented, multi-processor, time-shared computer system is set up. A drum system in particular is discussed, although no loss of generality is implied by limiting the discussion to drums. Particular attention is given to the queue of requests waiting for drum use. It is shown that a shortest access time first queue discipline is the most efficient, with the access time being defined as the time required for the drum to be positioned, and is measured from the finish of service of the last request to the beginning of the data transfer for the present request. A detailed study of the shortest access time queue is made, giving the minimum access time probability distribution, equations for the number in queue, and equations for the wait in the queue. Simulations were used to verify these equations; the results are discussed. Finally, a general Markov Model for Queues is discussed in an Appendix.

Thesis Supervisor: J. B. Dennis
Title: Associate Professor of Electrical Engineering

ACKNOWLEDGEMENT

I wish to express my appreciation to Project MAC for the use of its computational facilities, without which this paper would not have evolved; to Professor J.B. Dennis, my thesis advisor, for the many hours spent in discussing the ideas contained here; to A. L. Scherr for the use of his simulation language and for the many useful ideas that he gave; to Professor A. W. Drake, Professor D. C. Carroll, A. A. Smith, and E. Van Horn for constructive and enlightening criticisms; and to my wife Anne for her patience and encouragement.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

CHAPTER I. INTRODUCTION.

With the advent of more and more complex computing systems it has become increasingly important to have some reliable means for evaluating the performance of the system. In the Compatible Time-Shared System (CTSS) at Project MAC (2), M.I.T., for example, the scheduling of users is a problem that is receiving much attention. Patel (14) has considered first-come-first-served allocation of processor resources to users, and a multiple-level dynamic priority scheduling algorithm which closely models the scheduling algorithm used in CTSS (2). Heller (10), on the other hand, has considered the more general problem of a multiple-processor time-shared system. The purpose of the scheduling algorithm is to allocate the processor resources as efficiently and equitably as possible, minimizing processor idle time and user waiting time. Various schemes for scheduling have been tested at MAC but the one described by Patel has proved most satisfactory. Scherr (17) has made a far-reaching study of CTSS-like systems, with particular emphasis on their Markovian aspects.

Before the user's waiting time can be minimized it is necessary to minimize the processor idle time. One of the most inefficient operations is the swapping of information between the core memory and the drum or disc files. Oftentimes the processor must stand idle during a swap, awaiting the arrival in core of a block of data. One way to ease this difficulty is to use

one or more processors and let several programs occupy core at once. Then during the time that the swapping for one program is taking place, the processors can be kept busy on other probrams. In this way overall processor idle time can be reduced. These ideas of multiprogramming and multiple processors are not new; it is only recently that computer hardware has become sufficiently sophisticated to handle the task effectively.

Additional alleviation of the swapping problem can be effected by making drum and disc file operation as effieient as possible. In single-program systems efficiency of drum operation is not a problem since only one program (the program) can demand use of the drum at a time. Clearly, in a multiprogrammed system several programs can make simultaneous demands on the drum and disc facilities, making special organization a must to minimize the waiting time of a given program for its request to be serviced, and at the same time minimizing idle time of the entire system. It is clear that in a poorly organized drum system the inefficiency of the drum system can seriously impair the operation of the rest of the computing system because continued operation often depends on the reading of information into core: a program cannot begin to operate a segment until that segment has been placed in core. For instance, suppose we had at our disposal the means to reduce the average service time of a drum request by two or three milliseconds. In the two or three milliseconds

saved much computation can be performed.

In this paper we consider a model for a drum file memory system, and in particular a model for the programs in such a system. The model will describe the manner in which a program (or more properly, a <u>process</u>) makes requests for file memory use. A computer simulation has been written for the particular model described. In Chapter 4 a pertinent mathematical model is given. In Chapter 5 the results obtained from this model are compared with the results obtained from the simulation. The interested reader is referred to Scherr (17) and to Appendix 4 for an outline of the complexity of even the most tractable of models, the Markov Model.

CHAPTER 2. BACKGROUND.

It is the purpose of this section to discuss some of the concepts upon which this paper is based. One of the problems of existing time-shared systems is that the processor must stand idle while the present and next user's programs are being swapped in or out of core. One proposed solution to the problem is to run one user's program in core, meanwhile swapping the next user into a remaining part of core. Then the processor would be switched to the next user, and the swapping operation would begin anew. Of course each user would not be arbitrarily assigned half of core, but programs would be matched in some opmplementary manner long ones with short ones. This mode of overlapped operation in a time-shared system is sometimes referred to as a ring (cf. Scherr (17)). Again, idle processor problems arise if one program should require all available core space. Then no simultaneous swapping could take place.

A generalization of the above solution to the problem has been considered at length by J.B. Dennis and E Van Horn (3,6). It is known as segmentation. Under this scheme a user's program would be divided into a set of individually named parts, called segments. The user is assumed to have segmented his program in the way which seems most appropriate to him.

Segments may be classified roughly according to the manner in which they may be accessed:

(1) Read-Only.

(2) Data.

(3) Pure procedure.

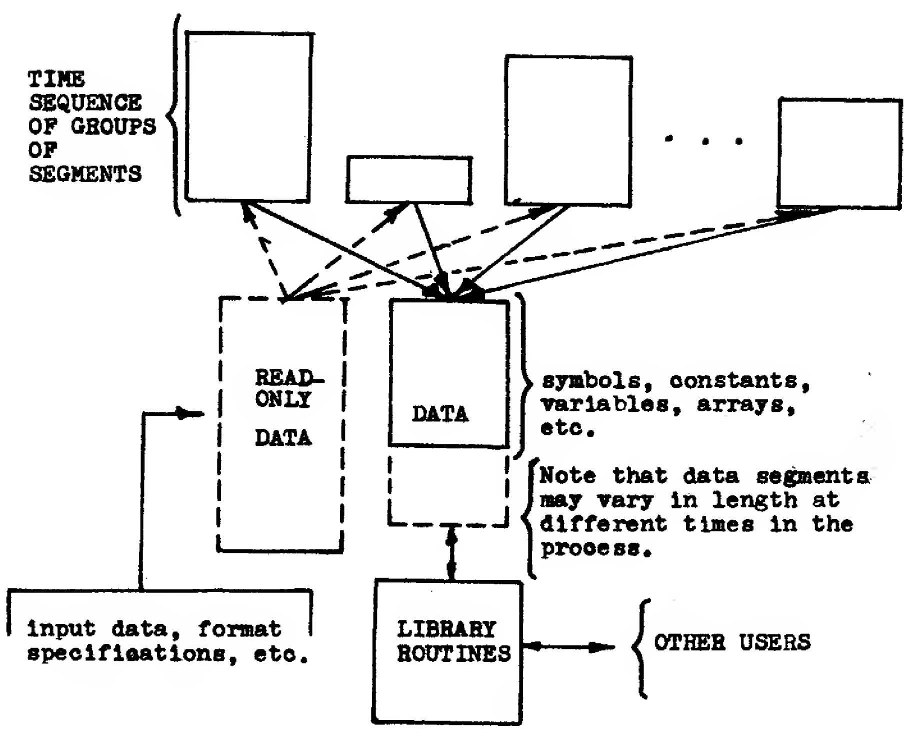
Some combinations of these classes are permitted.

A <u>pure procedure</u> segment is a set of instructions which directs a process* to operate on <u>data</u> but not on itself. Thus we could ask the compiler to extract all the symbols, variables and so forth, from a program and group them into one segment; procedure segments would then be allowed to modify and use this data. Of course certain programs, notably short ones, would be contained entirely in one segment. <u>Read-Only data</u> might be input data, format specifications, and so forth, which are not altered by the processes in a user's computation. Operation of a program might be in the following manner. Some first segment would be brought into core, together with all necessary data segments, which may or may not include read-only data. Then segments may act singly or in groups (if several processors are available) on the data. New segments are brought in as needed (when a reference is made to a segment not already in core). The programmer may wish to declare <u>subroutine segments</u>, which might contain some of

*A process is carried out by a processor under the direction of instructions in procedure segments. (3.4.7).

his often-used subroutines, and which for efficiency' sake should be kept handy in core at all times. Of course certain subroutines, such as printing or exponentiation subroutines, might be kept in special common, or library segments, being available for the common use of all users. In this way each user would not need to be given his own copy of each and every library routine. Figure 2.1 suggests the operation of the system, showing a time sequence of groups of segments operating on data. The time sequence may not be in the order in which the segments were written, and the same segment may appear many times in the sequence. Several processors might be available to work for one user, so that several segments might be active at once. Note that we have indicated that the segments are in general of various lengths. Note too that read-only data may not need to be present in core (main memory) but may be referenced from, say, a drum memory (auxiliary memory) as needed.

Clearly, by writing programs in segments, only a few segments of a given program need be in core at once, the rest being stored in auxiliary memory, perhaps on a high speed drum. A segment in core which is being used by one or more processes is called a working, or active segment. Segments kept on the drum are called dormant segments. Many users, or course, can have segments working simultaneously if there is more than one processor available. When a segment is working it can have one or more processes taking place in it, depending again on how many processors are available to work



<u>Figure 2.1</u>. Operation of one user in a segmented system.

on it. Hence when talking of the computations within a user's set of segments, we shall speak of a <u>user's processes</u> rather than a user's program.

Each segment will be named in some arbitrary manner. When a process makes reference to a segment (by <u>naming</u> it and giving the <u>address</u> of some <u>word</u> within it) which is not in core, that process is temporarily suspended until the required segment is brought into core. Since many users may have simultaneous processes it will be necessary to have some central control over allocation, swapping, and so forth. The program which does this job is called the <u>Supervisor</u> program. When a process references a segment other than the one in which it is taking place, the Supervisor will transfer control to that segment if it is in core. Otherwise that process must halt until the Supervisor has brought in the needed segment. With many processes running there will be a great demand for drum usage. We think of a process <u>causing</u> a request to be made to the drums for information, rather than the process itself making the request. We can see that references to other segments are at arbitrary points in time, and may be to arbitrary segments, which may have arbitrary length.

If requests should be generated momentarily faster than they can be serviced, then the waiting requests must be places in a waiting line, or queue. The order in which requests are serviced (i.e., the order in which they leave the queue) is not necessarily the order in which they arrived at the queue. We can see three distinct parts of the data transmission function of the computing system: the users' processes, which generate requests (either to read or to write on the drum); a queue into which requests that have to wait are placed, and which has a selection rule for next out, called the queue discipline; and finally the drums.

One final word must be said, concerning the transmission of data to and from the drums. It seems both desireable and convenient to have some standard unit of transmission and allocation, which we call the page. It is always possible to store pages consecutively on the drum (see Section 3.4). This requires that there exist some mechanism for deleting unnecessary data from the drums. One possible mechanism, using a percentage level of drum occupancy, is discussed in Section 3.4. It is necessary for the Supervisor to maintain some level of drum occupancy, and to have a

deletion policy in order to keep the drum from overflowing.

We will see under our study of queues in Section 3.3 and Chapter 4 that for each request there is a certain drum positioning time, or access time, that must pass while the requested starting page comes opposite the drum's read-write heads. This access time is wasted time. We seek to minimize it.

There are two general methods of handling core allocation, and it is not clear which method is more desireable. One method is called page-turning, the other segment-turning. Under both methods, a set of pages will be grouped as a segment and given a name. Under segment-turning a whole segment is brought into core and kept there at least until the various processes are finished with it. Under page-turning, one page of a segment at a time is brought in, and a new page is brought in only when needed. Under page-turning unneeded pages are deleted singly, while segment-turning deletes the entire set of pages belonging to a segment if any one of them is deletable. Page-turning seeks to minimize wasted core space; segment-turning seeks to minimize overall processing time per user. Each method has its advantages and disadvantages. There is some evidence that neither is better (cf Scherr's Thesis, where it is shown that the scheduling and computational time quanta do not significantly affect system operation (17)). This paper assumes a segment-turning system.

In conclusion: when a user's process refers to another segment that is not present in core, it will cause the Supervisor to generate a request to the drums. Ordinarily a request will be a read request, but it might also be a write request if the referenced segment is one in core being declared in the reference as "dormant"; or it may be a delete request if the referenced segment is being declared "dead". The queue will contain the waiting requests, while the drums will service them. A proper deletion policy is needed. Finally it is clear that the unit of information transmission ought to be the page, but the core memory allocation question, namely whether to allocate in pages or in segments, is open for discussion.

CHAPTER 3. THE DRUM SYSTEM.

3.1. Introduction.

The system model described here consists of three elements: the Users' Processes, the Queue, and the Drums. The Users' Processes element models requests to the Drums to read, write, or delete. The Processes will make requests at certain intervals given by some inter-request-time probability distribution; they will request some quantity of data in units of pages, beginning at a specified location on the drum. Several drums may be present, so each request will specify which drum is involved. Delete requests will be sent directly to the drums, while read and write requests will be entered in the Queue. The Queue will contain a list of which processes are requesting how much data from (or to) what drum, andthe starting location of the drum. It will act according to some queue discipline to decide which request is next to reach the drum, and will assign the request to a free channel to the requested drum. When a request is assigned to a channel it is deleted from the Queue. When a drum is notified by the Queue that a request is assigned to a channel it takes note of what program has been assigned to the channel, what the desired starting location and field are, and whether the request is a read or a write. A certain amount of time must elapse before the desired location has revolved into position; this time is the access time. Once the desired

starting position has come opposite the drum heads the data transfer begins, and ends after a certain amount of time, the transfer time, has elapsed. The sum of the access time and the transfer time is called the service time. The channel idle time is the time during which the channel has no request assigned to it. There may be some question whether access time should be included in channel idle time. Since access time directly affects a given request's wait before the end of its service, we have included it in the service time. Figure 3.1 shows the system in block diagram form, as we have just outlined it.

We now give a complete description of each element starting with the most basic, and most probabilistic, the Users' Processes.

3.2. The Users' Processes Model.

In order for proper control of all computing facilities to be maintained, the individual processes in core do not make requests directly to the queue and drums. As discussed in Chapter 2, a request originates from the Supervisor, the program which controls allocation and proper operation of the system facilities. The Supervisor can prevent interaction between processes, providing protection against such

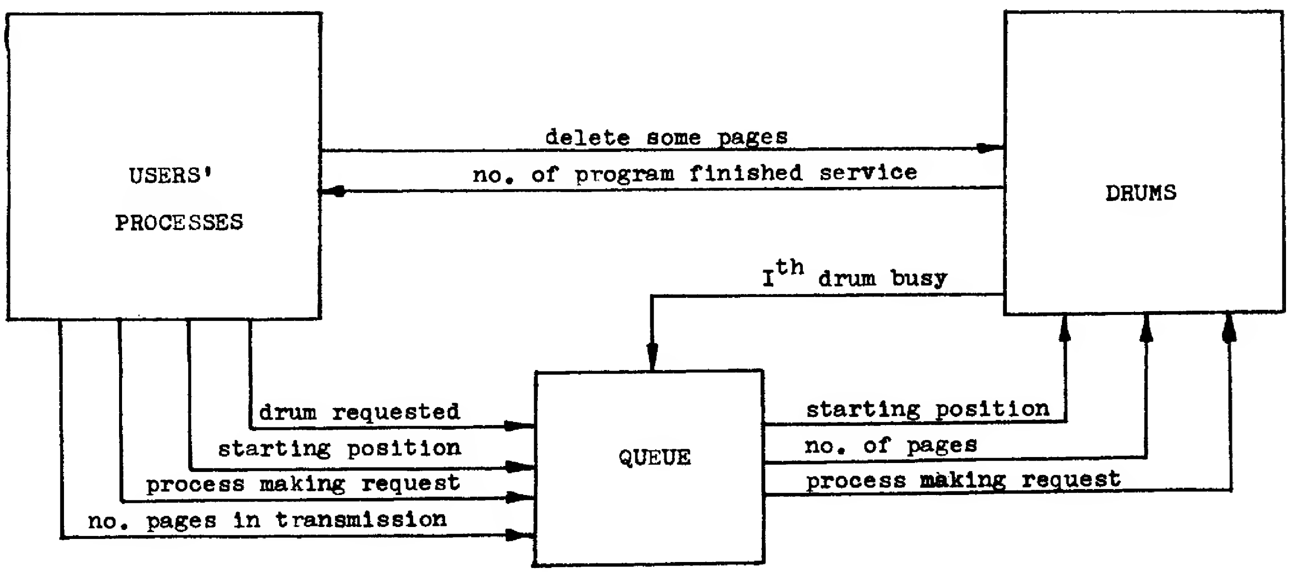


Figure 3.1. The Overall System Model.

happenings as some process erroneously requesting to write on top of another's information. The Supervisor will contain the queue.

In order to promote efficient operation, program segmentation will be used (3,6). By breaking the program into segments, efficient use can be made of core memory, since those segments of a program in which no processes are presently taking place should be stored on the drum and should not be "cluttering up" core. When a process references a segment not in core, the Supervisor will request that the next segment or segments be brought into core. Clearly while the next segment or segments are being read into core, any waiting processes are suspended; hence our first assumption:

> Assumption 1. Once a process has caused a request for one or more segments to be read in, it is temporarily syspended until its new segments are brought in. In particular a process will be unable to cause further requests until at least the time when it is resumed.

On the other hand, during the course of computation a process may generate some output data in core and request that this data segment be stored on the drum, for example so that it can reuse the same core space for further data. Such write requests do not imply that the process must come to a halt, hence our second assumption:

> Assumption 2. Upon generating a write request a process may continue, and in particular it may cause further read or write requests while a write request is being serviced.

From the above discussion, we may expect that a read request is more probable than a write request, and so our third assumption:

> Assumption 3. The probability of a process causing a read request is not the same as that of it causing a write request, and in general the probability of a read is greater than that of a write.

In order to simplify space allocation on the drums, the surface of the drum will be divided into blocks, or pages, consisting of some fixed number of words. Thus the number of words per page is fixed, and

> Assumption 4. The unit of information transmission and storage will be the page.

We have no reason to assume that the number of pages in an arbitrary segment is fixed; in fact all we can say is that long segments (those with many pages) will be unlikely as will extremely short segments (for example one or two pages). The number of segments in a block of n pages ia a random variable, and in particular the probability of finding exactly n pages in s segments may be given by a discrete Poisson Distribution:

$$P(s,n) = \frac{(n/\overline{N})^s}{s!}\, e^{-n/\overline{N}} \qquad \begin{matrix} n=0,1,2,\ldots \\ s=1,2,3,\ldots \end{matrix} \qquad (1)$$

where the mean number of pages per segment is $\overline{N}$.

Consider this problem: if a process should reference more than one segment not in core, so as in initiate the read-in of several segments, should the Supervisor ask for

the several segments in a single request, or should it make separate requests, one for each segment? We are assuming that it is always possible to store the pages of a given segment sequentially on the drum, that is that we can always read or write a segment without interrupting the transmission between start and finish. How this is done is considered in some detail in Section 3.4. For three reasons we argue that in the event of need of several segments contemporaneously there should be a separate request made, one for each segment. First, since consecutive segments may not be all written at once, but may have been written at widely spaced intervals, and independently or each other, it is unreasonable to assume that segments will always be stored consecutively; although this could be done by the method of Section 3.4. Second, there is no assurance that the requested segments will all be on the same drum, or that the request will even be for consecutive segments. Finally some queue disciplines discriminate against long requests, servicing those requiring the shorted service times first (Section 3.3); asking for several segments in one request could well result in an inordinately long wait for service under such a queue discipline. We now make our fifth and sixth assumptions.

> Assumption 5. Each request will be for one segment, but at a request time a process may cause several requests. The probability that s segments will be requested will be exponential, that is
>
> $$P(s) = e^{-s} \qquad s=1,2,\ldots \qquad (2)$$

Furthermore at request time there is no reason for all the requests to be either all read or all write; they may be mixed, A read request, or course, will cause suspension of the process.

Assumption 6. The number of pages in the single segment of each request will have probability of being n pages

$$P(n) = \frac{n}{\bar{N}^2} e^{-n/\bar{N}} \qquad n=0,1,2,\ldots \quad (3)$$

where $\bar{N}$ is the mean.

When a segment is active, that is, when processes are referencing it, the probability that the next requests occur at each successive time instant are independent so that we expect the arrival times or requests to be Poisson Distributed. A request is unlikely to be made immediately after resumption of a process from the last request, and it is unlikely to be made an extremely long time after the resumption of a process. The probability of exactly k requests in a time interval t is

$$P(k,t) = \frac{(at)^k}{k!} e^{-at} \qquad t \geq 0 \quad (4)$$

where a is the average number of arrivals per unit time. We have then

Assumption 7. The inter-request times are taken from the following distribution*

$$P(t) = ae^{-at} \qquad t \geq 0 \quad (5)$$

---

*See page 60.

Something must be said about the starting position of the drum a particular request will seek. We have no information to allow us to assume anything other than that all drum positions are equally likely to be requested.

<u>Assumption</u> <u>8</u>. At a particular request time all drum positions are equally likely to be selected; that is, the density of angular positions requested will be

$$P(\theta) = \frac{1}{2\pi} \qquad 0 \leq \theta \leq 2\pi \qquad (6)$$

Finally something must be said about which drum is to be requested, in the event that there are several drums in the system. When a process is making requests for several segments there is no reason to assume that all the requested segments will be on the same drum. Hence we are willing to say that each of the D drums is equally likely to be requested:

<u>Assumption</u> <u>9</u>. Each request is equally likely to be for any of the drums in the system.

Assumptions 3,5,7,8, and 9 are illustrated in Figures 3.2 to 3.7.

Based on the discussion above, we are in a position to construct a model for the request activity of a given process. This model is shown in Figure 8.*

*A Note on Notation: A <u>fork</u> is a point at which one process splits into two processes, which follow their own paths. A <u>join</u> is just the opposite, where two processes become one; each time the join is entered the operations in the box of the flow chart are carried out. An arrow doing this ——▶| is a termination of a process. A note in brackets gives the condition permitting a process to emerge from the corresponding box. A function written with an argument (.) denotes a probability function for a set of identically distributed random variables.

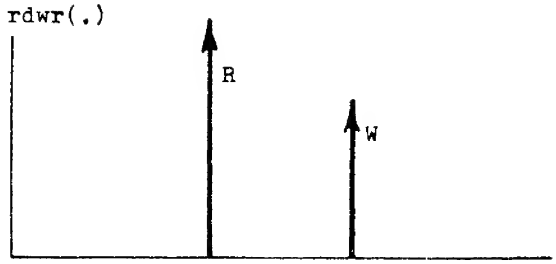


Figure 3.2. Relative frequencies of read and write requests.

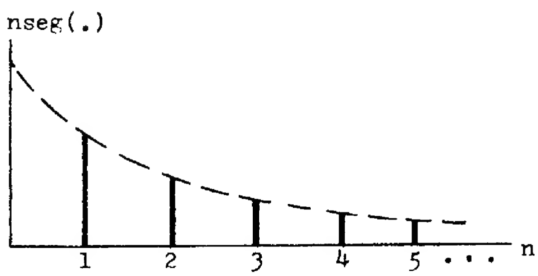


Figure 3.3. Relative probabilities of number of segments per request.

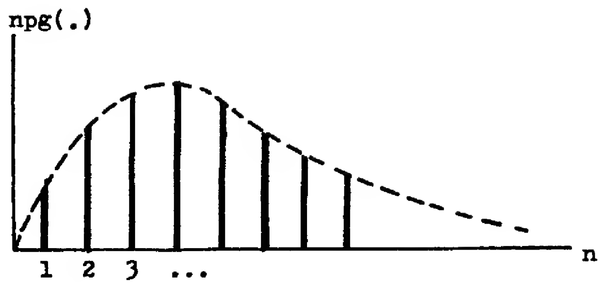


Figure 3.4. Relative probabilities of number of pages per segment.

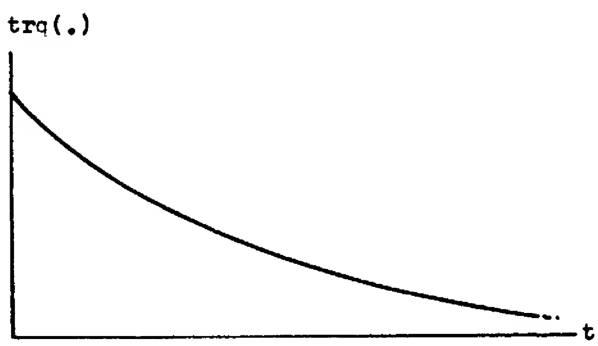


Figure 3.5. Relative probability of inter-request times.

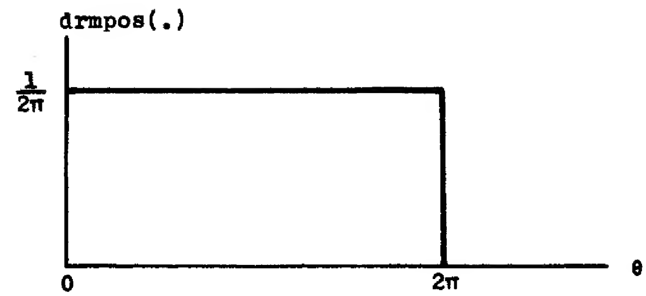


Figure 3.6. Relative probability of requested drum position.

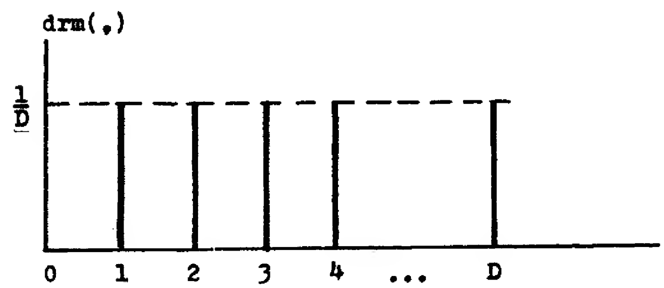
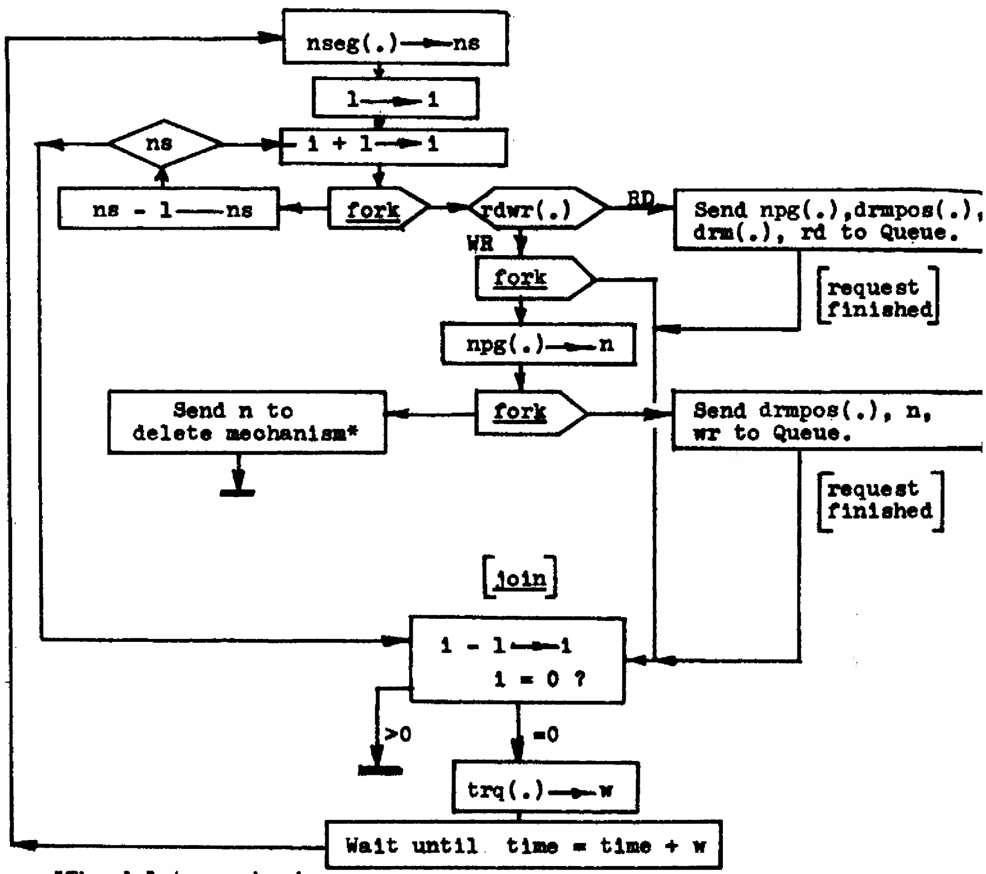


Figure 3.7. Relative probability of requested drum.



*The delete mechanism is discussed in Section 3.4.

nseg(.) = no. of segments requested
rdwr(.) = rd-wr distribution
npg(.) = no. pages requested
drm(.) = requested drum
drmpos(.) = requested drum position
n = no. pages this request
ns = temporary segment count

Figure 3.8. The Processes Model.

The reader may be asking what justification there is for assuming the particular probability distributions that have been chosen; in particular why we have chosen Poisson distributions as opposed to other distributions. It will be noted that these choices are completely arbitrary, and cannot be properly determined until some statistics are available about the system we are discussing. It is felt that the assumptions that have been made are reasonable.

3.3. The Queue.

The model of the queue is more straightforward and deterministic than the model of the processes. When a request is received from a process it is entered in a list within the Queue Element. Each entry in the list contains the following information: an identification number of the process requesting, the number of pages involved in the transmission, the desired starting location on the drum, the identification number of the desired drum, and whether the request is a read or a write. The number of pages is an important piece of information since it can be used to determine when the transmission is ended.

A possible structure for the Queue's list, which we will refer to simply as the queue, is shown in Figure 3.9. In this list two pointers are used, one to indicate the lower limit of the number in the queue (the shaded region), the other to indicate the upper limit. Both pointers are periodically incremented and are modulo capacity of queue. The lower pointer is moved down one position each time a new entry is made, and the upper pointer is moved down one position each time a request leaves the queue. If the next out is not the least recent entry, then all items above are moved down one position to fill the gap. The shaded area represents the number in the queue, frequently referred to as the length of the queue.

There is a Boolean signal received from each of the drums indicating whether or not that drum is busy (all channels to it in use). Whenever all channels to a drum are busy, any requests arriving for that drum must wait in line, and a waiting line, or queue, is formed. If requests arrive too much faster than they can be serviced, the length of queue could become equal to its capacity and any further requests will be lost. Such a development is disastrous, since it would render a process useless. Hence the average arrival rate must not exceed the average service rate, where the rates are defined to be the reciprocals of the average interarrival and service times, respectively.

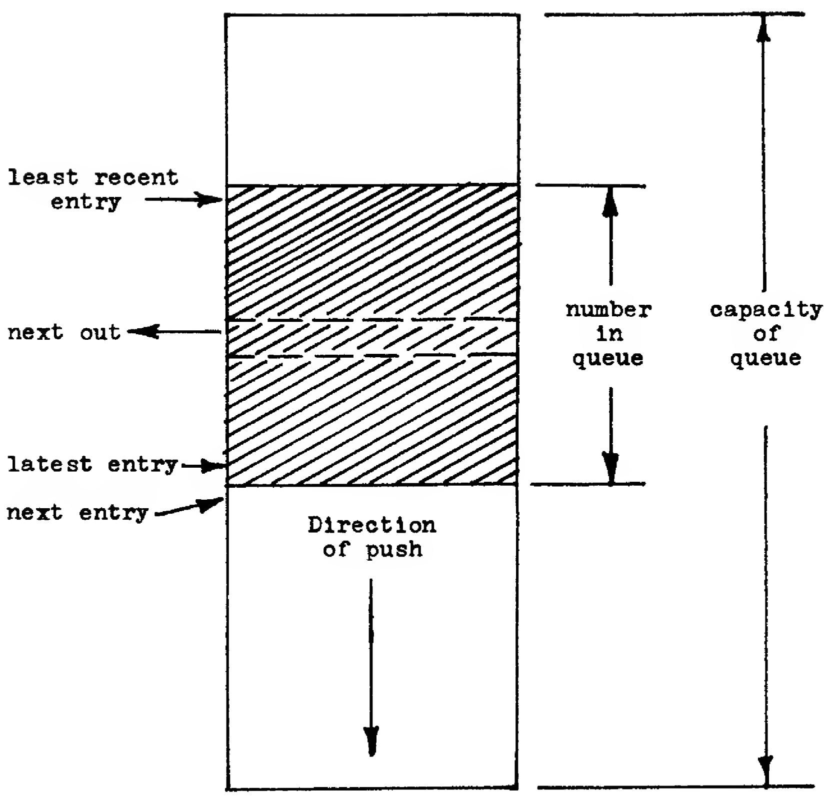


Figure 3.9. A structure for the queue stack.

When the Queue is aware that a drum is not busy, it looks down the list to determine which if any requests want the free drum. It then chooses one of them according to the queue discipline, assigns it to some free channel to that drum, the deletes the entry from the list.

The queue discipline is simply the <u>rule for selection of next out</u>. We consider four queue disciplines applicable to our situation:

(1) First come, first served.

(2) Shortest access time first.

(3) Shortest Job first.

(4) Mixed policy.

<u>(1) First come first served</u>.

This is the "fair" or "equitable" queue discipline, where requests are serviced in the order of their arrival, and is the case when the "next out" of Figure 3.9 is the "latest entry". It does <u>not</u> result in the most efficient operation. It is analogous to the normal situation encountered in a post office, when one wishing to buy a single stamp must wait behind a person with several packages. Certainly the waiting time is greatly increased because of ill fortune, whereas the person ahead would not be significantly delayed to give way. Since a process is equally likely to ask for any drum position, and since the present drum position is likely to be anything, with the first come first served queue discipline the average access time is half the drum revolution

time. Let us represent the time a request is in the service system (the time from when a process makes a request until the time service is completed) by T_s. Let the drum revolution time be T. Let the average transfer time be T_t. And let the average wait in the queue be W_q. Then for first come first served,

$$T_s = W_q + T_t + T/2 \qquad (1)$$

<u>(2) Shortest Access Time First</u>.

Under this queue discipline the next out is selected according to following rule:

> Choose the one for which the rotational positioning delay until the desired starting address is minimum.

Now if more than one request for a given drum is in the queue, on the average the access time will be less than half the drum revolution time; this is so since with more in the queue the probability that there is a request for the present drum position is greater than for a queue of length one. It will be shown later that the minimum access time is roughly inversely proportional fo the length of the queue. Hence for this queue discipline

$$T_s \approx W_q + T_t + T/\bar{n} \qquad (2)$$

where $\bar{n}$ is the average number in the queue, and W_q is not the same numberically as for the first come first served queue with the same $\bar{n}$. Observe that the shortest access time queue is a dynamic priority queue, one for which the priorities of requests are changing randomly.

(3) Shortest Job First.

Under this queue discipline the following rule is used to select the next out:

> Select the request for which the service time is a minimum. The service time is the sum of the access time and the transfer time.

A little thought should convince the reader that under this queue discipline the access time is not minimized, but yet it will in general be less than T/2 for queues of length two or more. Hence

$$T_s = W_q + T_t + T' \tag{3}$$

where $T/\bar{n} < T' < T/2$, and W_q is not the same numerically as for either a first come first served or shortest access queue having the same length $\bar{n}$.

(4) Mixed Policy Queues.

It will be noted that the shortest access time queue and the shortest job first queue are queues in which a continuous number of priorities exist. Suppose we become concerned about requests which might have to wait an inordinately long time, perhaps because of ill fate, perhaps because its job time is long. This could be a real problem in the shortest job first case (what of the longest job of all?). It ought not to be too much of a problem in the shortest access time case, since each time a request is to leave the queue, it has an equal chance among all the others of being chosen. This is only partially true for the Shortest Job First Case, where the job time has a random component, the access time; and if the transfer time component be very long, then the job time depends almost entirely on the transfer time. Notice that for very short transfer times, the shortest job queue will approach in operation the shortest access queue. One way of circumventing the problem of some request waiting inordinately long is to introduce a skip limit into our model. Each time a request is skipped over as next out, a counter associated with that request is incremented. If this skip count ever exceeds the skip limit then this request is next. What we have done in essence is to add a first come first served component to the queue. As the skip limit is lowered a shortest access time or shortest job queue behaves more and more like a first come first served queue. In fact a queue with the skip limit set to zero is just a first come first served queue.

We will include no more discussion on mixed policy queues since the problem is in general complex and unsolved. We will, however, mention the skip limit once again in Chapter 5 under the discussion of simulation results. Finally, another mixed policy queue is discussed in Appendix 3. For further discussions on the matter, the reader is referred to the literature (1,9,13,15,16).

(5) A Comparison of Queue Disciplines.

In Appendix 1 we have related the mean number in the service system, which includes those being serviced and those in the line, to the mean and variance of the service time distribution. We will denote the number in the system by L. The random variable of the service time, t_s, is the access time, t_a, plus the transfer time, t_t. The service distribution can be found from a convolution of the access distribution with the transfer time distribution. We will show later that both of these can be found, hence the service distribution can be found. In particular, the mean service time, T_s, is

$$T_s = T_a + T_t \qquad (4)$$

And the variance of the service time, σ_s^2, is

$$\sigma_s^2 = \sigma_t^2 + \sigma_a^2 \qquad (5)$$

since we are assuming independence of t_a and t_t. Let us denote the function relating L, T_s, and σ_s^2 by

$$L = F(T_s, \sigma_s^2) = F(T_a + T_t,\ \sigma_a^2 + \sigma_t^2) \qquad (6)$$

The function F from equation (6) is such that a decrease in either or both of T_s and σ_s^2 will result in a reduction in L. The transfer time distribution will remain the same for all queue disciplines since it is a function of the number of pages per segment, which is fixed before hand, and is assumed to be identical for all processes.

Let the total number of processes in all be N. Then

$$N = w + L \tag{7}$$

where w is the number of working processes. The efficiency of a system can be measured crudely by the number of processors working, and is

$$\text{efficiency} = \frac{w}{N} = \frac{N - L}{N} = 1 - \frac{L}{N} \tag{8}$$

To maximize the efficiency, L must be minimized. Thus the optimum queue discipline is the one for which is minimized. Notice further that the number of processors that can be kept working is just the number of working processrs:

$$\text{number of busy processors} = w = N(1 - \frac{L}{N}) \tag{9}$$

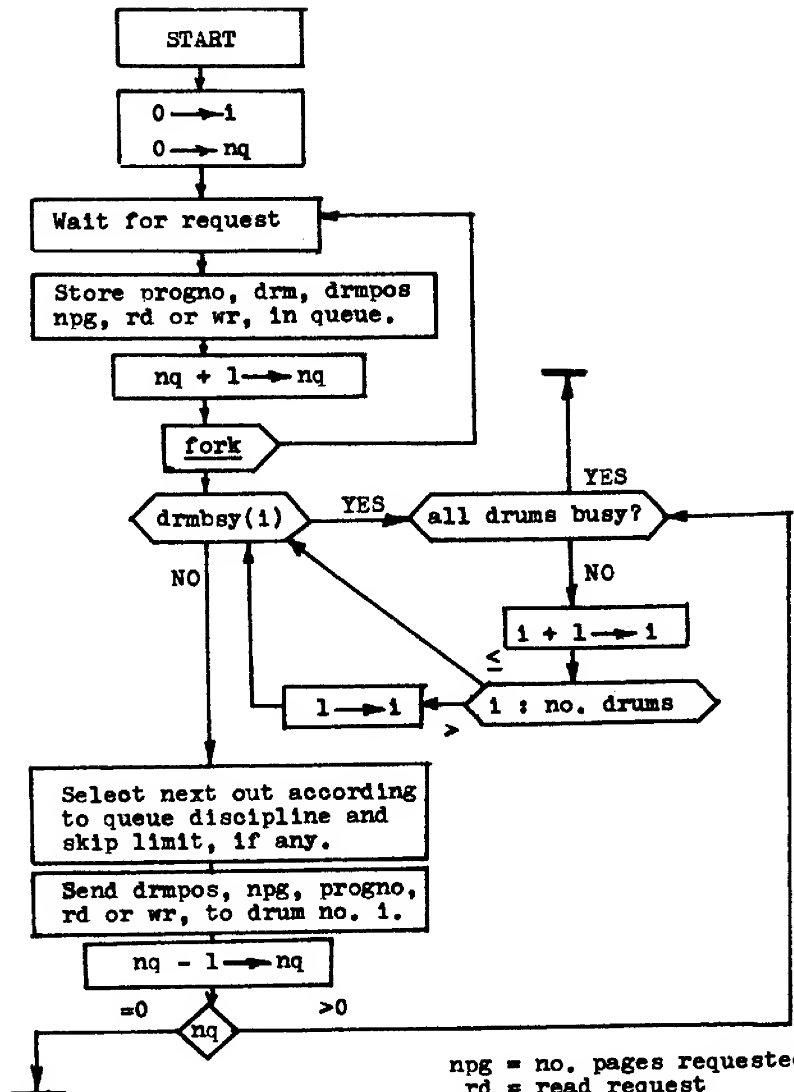
For the simplest system, the single-processor system, w must never be less than one if the processor is to be continuously busy.

On the average the transfer time is the same for all queue disciplines (because it relates directly to the number of pages in a segment). On the average the access time is explicitly minimized only by the shortest access time queue. Therefore the service time for the shortest access queue will, on the average, be a minimum, compared to other queues.

We see that the shortest access time queue minimizes the service time, while the queue which bears the "shortest job first" does not minimize the average service time. The apparent contradiction is resolved when we realize that the shortest access queue chooses the shortest job first <u>on the average</u>, while the "shortest job first" queue selects the shortest <u>instantaneous</u> job. Nevertheless equation (6) tells us that the shortest access queue must have minimal L associated with it, and is therefore the most efficient*. In fact, any queue which does not minimize the access time must be less efficient than a shortest access time queue, when efficiency is defined by equation (8). Chapter 4 is devoted to a detailed study of this queue.

Based on the above discussion, we give the Model of the Queue in Figure 3.10.

*This conclusion is verified by simulation. See Chapter 5.



npg = no. pages requested
rd = read request
wr = write request
drm = the drum desired
drmpos = the starting position
drmbsy(i) = indicates ith drum busy
nq= number in queue

Figure 3.10. The Queue Model.

3.4. The Drums.

Since the method of distributing pages on the drum is of considerable importance, we will discuss it first. Consider Figure 3.11. The drum, we suppose, is divided into sectors as viewed from a cross-section, where the number of sectors is an integer. The number of words per page is just the number of words that can be written around the circumference of the drum divided by the number of sectors. The drum is divided into rings, and the width of one such ring is a field. A field is one word in width, and a word is typically 36 binary bits. Each field is subdivided into a number of tracks, each of which is associated with one read-write head. The same head is used for reading and for writing: a read amplifier or write amplifier is connected as needed. The operation a head is presently performing is called its status, and there is a delay associated with switching between read and write status. This selection delay is about the same time as for three or four words to pass beneath the head, so ordinarily the first few words on a page will be left blank to allow for this delay.

It was stated previously that it is possible to write a segment of N pages on the drum contiguously. We indicate how this can be done. The question is: suppose some of the sectors in a field are used, how can a string of N pages be written consecutively, especially if a page would have

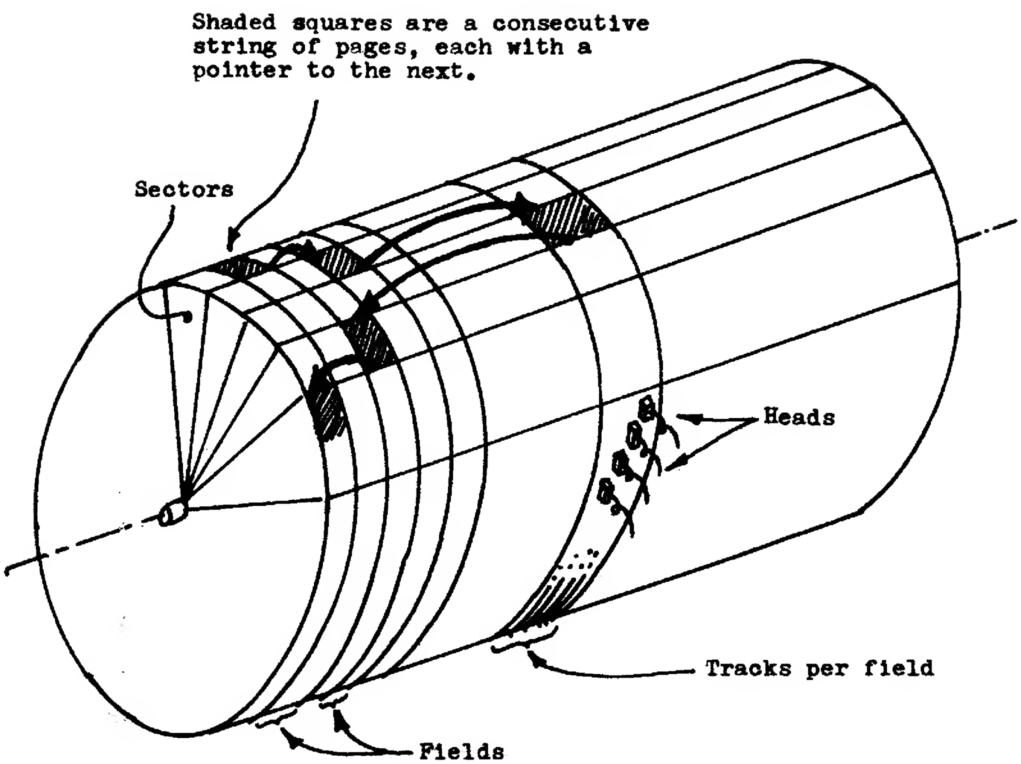


Figure 3.11. Organization of the drum.

to be written on a used sector? The answer is that we do not attempt to write the pages in the same field. We require only that during a write operation there be at least one free field per sector. First of all, suppose that each sector was allowed to have all but C of its fields in use, where C is the number of channels to the drum, and where any channel can access any field. Suppose further that whenever fewer than C fields were free on a given sector a deletion occurred immediately. Then the drum could handle C simultaneous write requests because a free field can always be found. In reality a delete might not occur when necessary, and also there is the possibility that a segment is longer than the number of sectors, which implies that more than one of its pages would be written on the same sector. It would be better to set a <u>drum occupancy level</u>, which is the ratio of allowed fields per sector to the actual number of existing fields per sector:

$$\text{occupancy level} \leq \frac{F - C}{F} \qquad (1)$$

where F is the number of fields per sector, C is the number of channels to the drum. Then whenever the occupancy level is exceeded, some sort of emergency condition would be set up, and any unnecessary segments would be removed from the drum (they would be deleted, or they might be moved to a lower level of storage, for example a disc file). In such a case the occupancy level would have to be less than the upper bound set by the equality sign in (1) to allow for statistical fluctuations. Simulation has shown that

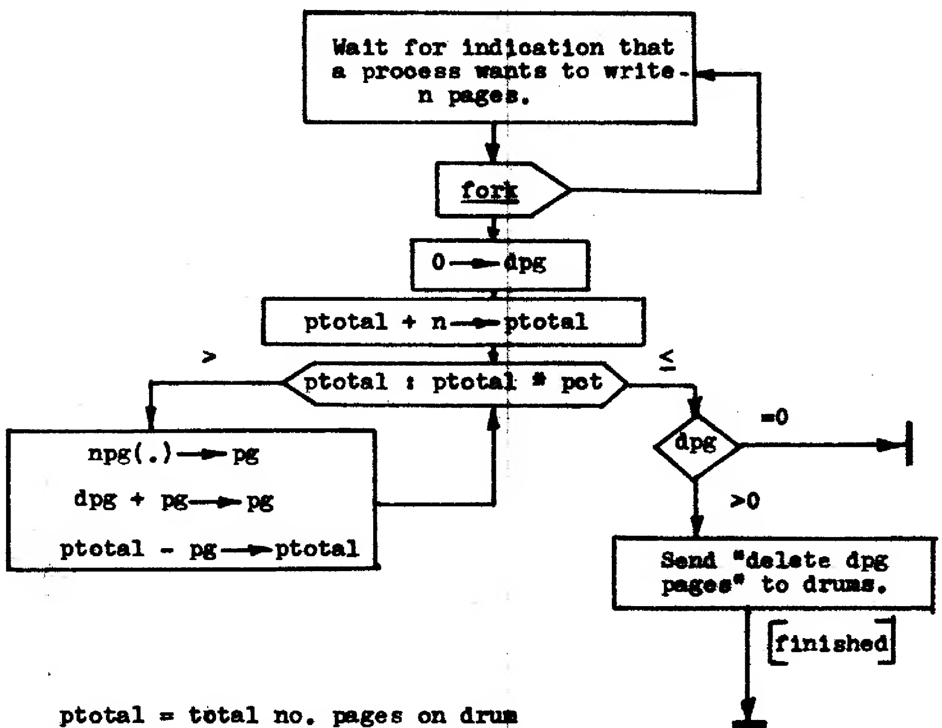
occupancy levels in excess of 93% are possible with a proper deletion policy, for typical parameters. See Chapter 5.

When a write request comes, the pages are written on the drum on the first free field on each sector, and a pointer is left, to direct a read operation to the next field of a consecutive string of pages. These pointers are indicated by arrows in Figure 3.11. Thus it is possible to have a string of consecutive pages written (and read) without interruption. We require rapid inter-field switching, a feature available on high-speed drums. It is to be noted that if the drum is be be operated this way it will have to maintain its own "Field Usage Table" similar in principle to the "Track Usage Table" used in CTSS with the disc(2). When a write request arrives, this table is consulted to locate the nearest free field on the given sector.

As long as the Supervisor's deleteion policy sees to it there are always sufficient free fields on each sector, the drum operation is straightforward. The delete mechanism shown in Figure 3.12 determines how many pages are to be deleted from the drum; it does this whenever the desired occupancy level is exceeded. We may model the behavior of a deletionn by picking a random drum address and deleting one page from each sector until N pages are deleted. A "deletion" may be to remove the offending pages to a lower level of storage, or it may be to obliterate the pages entirely.

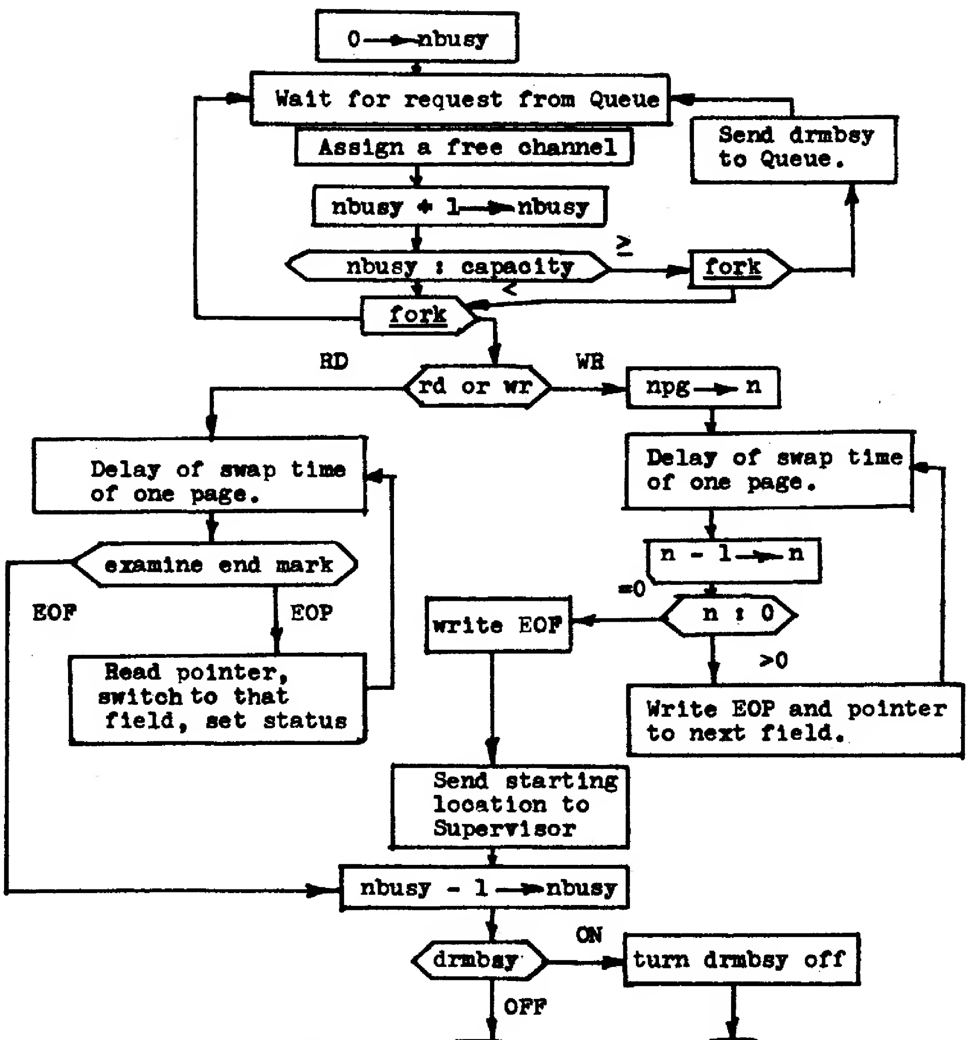
Once a channel is assigned, the drum observes whether the request is a read or a write, and switches the heads associated with that field to that status, as soon as the starting sector is apposite the heads. Note that the set of heads associated with a given field may be in use by different requests from sector to sector. When the starting page is in position, the data transfer begins, allowing time for the switching delay at the top of each page. Three or four words left blank on a page is sufficient time for this. At the bottom of each page is an End of Page mark, with a pointer to to the field containing the next page, which initiates switching to that field; of course the heads there are put into the proper status. Finally, at the end of the last page of thesegment, an End of File mark will be encountered, and the channel is freed for the next request.

The ideas for the drum model are embodied in Figure 3.13.



ptotal = total no. pages on drum
pct = per cent drum occupancy
dpg = no. pages to be deleted
pg = temporary page count

Figure 3.12. The delete mechanism.



capacity = no. of channels
nbusy = no. busy channels
npg = no. of pages
drmbsy = indicator that drum busy

<u>Figure 3.13</u>. The Drums Model.

CHAPTER 4. THE SHORTEST ACCESS TIME QUEUE.

4.1. Introduction.

In Section 3.3 it was shown that the shortest access time queue discipline is the most efficient; it is the purpose of Sections 4.2 and 4.3 to analyze this queue as best can be done. The shortest access time queue is a special form of the shortest job first queue. Solutions have been obtained for shortest job first queues with the input rates independent of the queue length. No solutions have been obtained for a shortest job queue in which the input rate is dependent on the queue length, that is, when there is only a finite number of requestors. In the next two sections we do not attempt to solve for the probability densities of queue length, waiting times, and service times; rather we talk only of the averages, which become time independent at equilibrium, when the input rate to the system is the same as the output rate from the system. In Section 4.2 we derive a probability density function for the minimum access time as a function of the mean number in the queue; then in Section 4.3 we combine these results with the results of Appendix 1 to obtain some approximate expressions for the number in the queue, and for the waiting time in the queue.

4.2. <u>The Minimum Access Time Distribution</u>.

The access time is defined as the time from the exit of a request from the queue until the requested starting sector has come opposite the read-write heads. It is simply a positioning time. We have shown in Section 3.3.5 that a queue discipline which minimizes the access time is the most efficient; we wish to derive the access time distribution in this section, and in a later section we will determine the waiting times in queue using the Pollaczek-Khintchine Formula (Appendix 1). We define the following quantities, given that n are in the queue:

R_i = requested starting sector of the drum for the i^{th} request in the queue.

$D(t)$ = The angular drum position at time t.

$A_i(t)$ = required access time at time t for the i^{th} request given that the present drum position is $D(t)$.

a = random variable of minimum access time, which takes on values a_o.

T = drum revolution time.

The model of the shortest access time queue discipline shown in figure 4.2.1 best illustrates what is going on.

The comparators compare the requested starting sector with the present drum position and give as an output the required access time. The Min(.) box selects the minimum of its inputs and sets its output to this value. To simplify the derivation we will assume that the Min(.) box normalizes its output with respect to the drum revolution time T, so

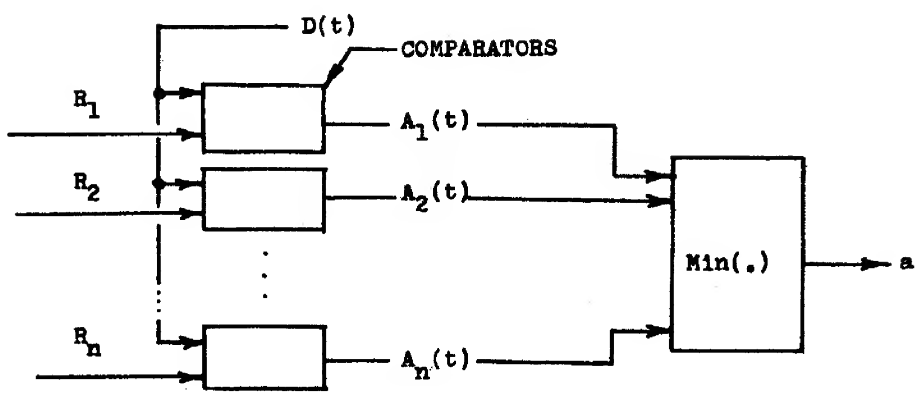


Figure 4.2.1. Operation of shortest access queue.

that a is a fraction between 0 and 1. We have

$$a = \frac{1}{T} \text{Min} \left[A_1(t), A_2(t), \ldots, A_n(t)\right] \qquad (1)$$

where $0 \leq a \leq 1$. We are interested in the probability density of a as a function of n, the number in the queue.

It was stated in Section 3.1 that the probability density of the R_i is uniform, that is, all drum sectors are equally likely to be requested. Further we are assuming random segment lengths. If segment lengths and starting positions are random, the present drum position, which is the drum position just at the finish of the last request (so that the next request is about to be assigned), is random, and by symmetry and the independence of requests, we may assume that it is uniformly distributed.*

*This is not true in the case of short segments because the drum will have rotated only a short distance. This matter is discussed further in Section 4.2, page 63.

But if $D(t)$ and R_i for each i are uniform, then $A_i(t)$ must be uniform for each i; that is, the i^{th} request's access time is equally likely to be any fraction of a drum revolution. Figure 4.2.2 shows the density function for $A_i(t)$, which has been normalized with respect to the drum revolution time, T.

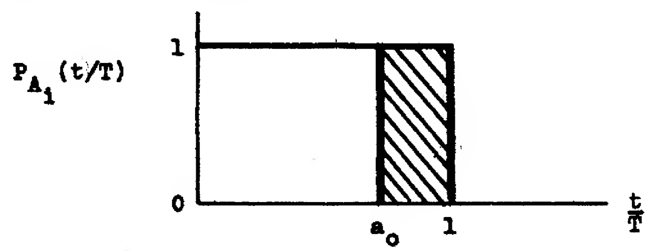


Figure 4.2.2. Access time for i^{th} request.

Now, the probability that $a > a_o$ is just

$$P[a > a_o] = P[A_1(t')>a_o,\ldots,A_n(t')>a_o] \qquad t' = \frac{t}{T}$$

But the R_i are independent, so that the $A_i(t/T)$ are also independent, and

$$P[a > a_o] = P[A_1(t')>a_o]\ldots P[A_n(t')>a_o] \qquad t' = \frac{t}{T}$$

But $P[A_i(t')>a_o]$ is just the shaded portion of Figure 4.2.2, and is simply $(1 - a_o)$. Then

$$P[a > a_o] = (1 - a_o)^n \qquad (2)$$

equivalently $P[a \leq a_o] = 1 - (1 - a_o)^n$

and $$P_a(a_o) = \frac{d}{da_o} P[a \leq a_o]$$

$$P_a(a_o) = n(1 - a_o)^{n-1} \qquad (3)$$

Equation (3) is the probability density of a, given that n are in the queue. Figure 4.2.3 shows $P_a(a_o)$ for a few values of n.

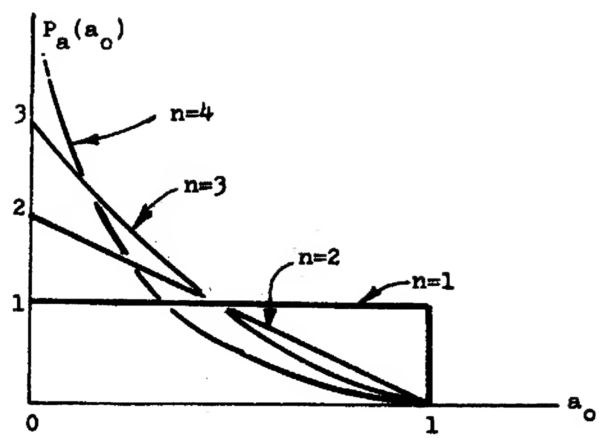


Figure 4.2.3. Shortest Access Time Distribution.

By the definition of conditional probability:

$$P_{aN}(a_o,n) = P_{a/N}(a_o/n)P_N(n)$$

where N is the random variable of the number in the queue, which takes on values n. Then

$$P_{aN}(a_o,n) = n(1 - a_o)^{n-1} P_N(n)$$

The mean access time, $\bar{a}$, is

$$\bar{a} = \sum_{n=1}^{\infty} \int_0^1 a_o n(1 - a_o)^{n-1} P_N(n)\, da_o$$

Integration by parts over a_o leads to

$$\bar{a} = \sum_{n=1}^{\infty} \frac{1}{n+1} P_N(n) \tag{5}$$

The second moment, $\overline{a^2}$, is

$$\overline{a^2} = \sum_{n=1}^{\infty} \int_0^1 a_o^2 n(1 - a_o)^{n-1} P_N(n) da_o$$

Integration by parts over a_o leads to

$$\overline{a^2} = \sum_{n=1}^{\infty} \frac{2}{n+1} \frac{1}{n+2} P_N(n) \tag{6}$$

The variance of the access time distribution is then

$$\sigma_a^2 = \overline{a^2} - \bar{a}^2 = \sum_{n=1}^{\infty} \frac{2}{(n+1)(n+2)} P_N(n) - \left[\sum_{n=1}^{\infty} \frac{1}{n+1} P_N(n)\right]^2 \tag{7}$$

Note that we have not specified $P_N(n)$, the distribution of the number in queue. Note too that it is not the same as the time distribution of n. It is the distribution of number in queue as seen by the departing requests--we need $P_N(n)$ taken over instants when the next request is extracted from the queue, which does not happen at uniform intervals. It is a reasonable assumption* that $P_N(n)$ is a normal distribution. This is only an approximation, since the normal distribution would allow for some probability of negative n, which is physically meaningless; this must be used carefully when

*Based on the Central Limit Theorem.

n is small enough so that the portion of the normal curve extending below n=0 is appreciable, especially when the variance of $P_N(n)$, σ_n^2, is large, so that $\sigma_n \gg \bar{N}$. $P_N(n)$ is

$$P_N(n) = \frac{1}{\sqrt{2\pi}\sigma_n} \exp[-\tfrac{1}{2}(n - \bar{n})^2/\sigma_n^2] \qquad (8)$$

Putting (8) into (5),

$$\bar{a} = \sum_{n=1}^{\infty} \frac{1}{\sqrt{2\pi}\sigma_n} \frac{1}{(n+1)} \exp[-\tfrac{1}{2}(n - \bar{n})^2/\sigma_n^2] \qquad (9)$$

And putting (8) into (7),

$$\sigma_a^2 = \sum_{n=1}^{\infty} \frac{1}{\sqrt{2\pi}\sigma_n} \frac{2}{(n+1)(n+2)} \exp[-\tfrac{1}{2}(n - \bar{n})^2/\sigma_n^2] \qquad (10)$$

Equations (9) and (1) cannot be reduced further, even if the summations are taken to be integrations over the infinite interval. These equations do, however, yield readily to a computer, and families of curves for $\bar{a}$ and σ_a^2 have been assembled and are shown in Figure 4.2.4 and 4.2.5. The axes are normalized so that, given the drum revolution time T, values of access time can be found.

We wish to note the limiting forms of equations (9) and (10). These occur for $\bar{n} \gg 1$, and for $\sigma_n \ll \bar{n}$. Figures 4.2.4 and 4.2.5 show that for $\bar{n} \geq 8$ we may ignore the effects of σ_n, for σ_n of interest (see Section 5.2), with only a small error. Now if σ_n is very small compared to $\bar{n}$ then the normal curve approaches a unit impulse in the limit,

and the summations of equations (9) and (10) reduce to a single value, taken at $n = \bar{n}$. Thus for $\sigma_n \ll \bar{n}$:

$$\bar{a} = \frac{1}{n+1} \qquad \text{at } n = \bar{n} \tag{11}$$

$$\sigma_a = \frac{1}{n+1} \sqrt{n/(n+2)} \quad \text{at } n = \bar{n} \tag{12}$$

It is to be noted that (11) and (12) are evaluated at $n = \bar{n}$, and that the approximation is very good if the conditions are met; this is evidenced in Figures 4.2.4 and 4.2.5, where equations (11) and (12) have been drawn. One of the prime assumptions of this derivation is that the drum positions at successive request-granting times are independent. If the drum positions at successive request-granting times are not independent, then the access distribution is in error. This is the case if the average length of requests is small compared to a drum revolution. See the discussion on page 63.

In Appendix 2 one further result of interest is obtained. The form of the probability density for the waiting time in queue is derived and is shown to be exponential. This is in excellent agreement with the simulation results discussed in Section 5.2.

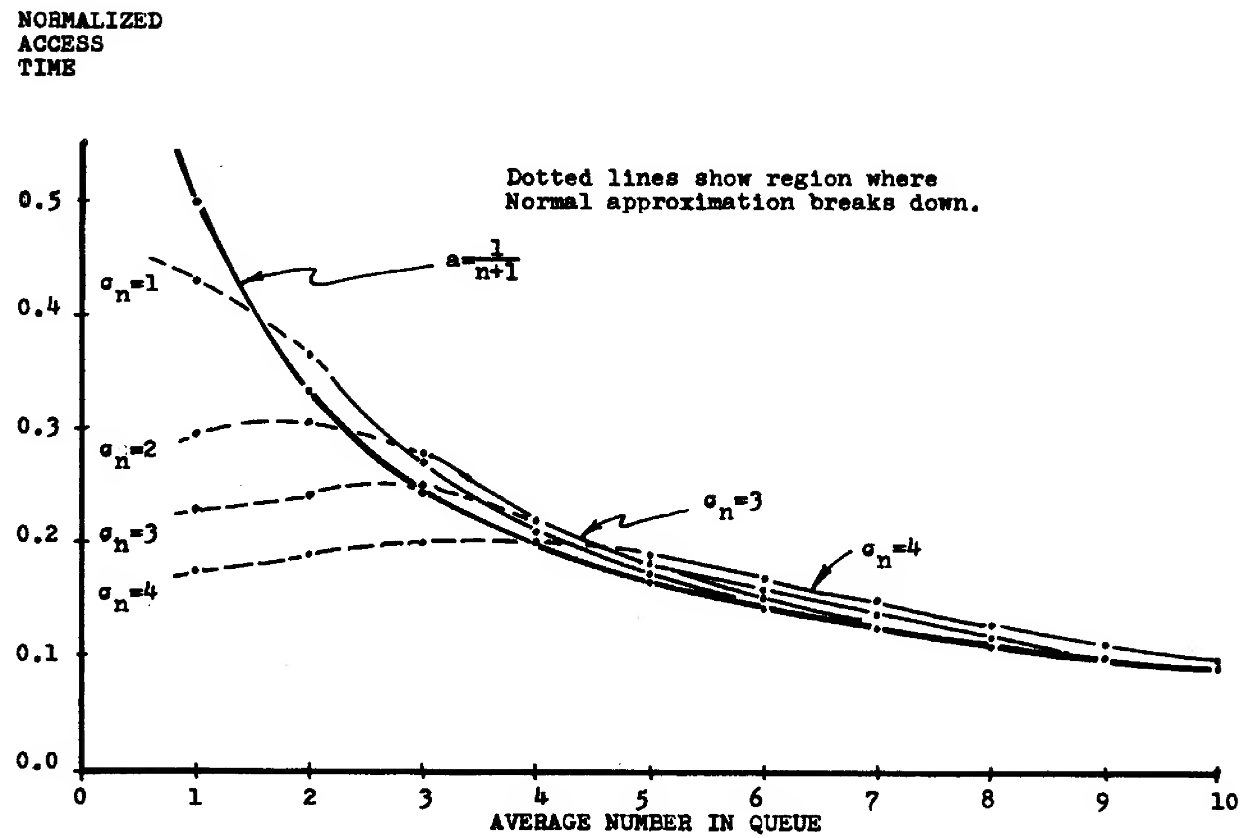


Figure 4.2.4. Access time normalized w.r.t. drum revolution time, as a function of average number in queue.

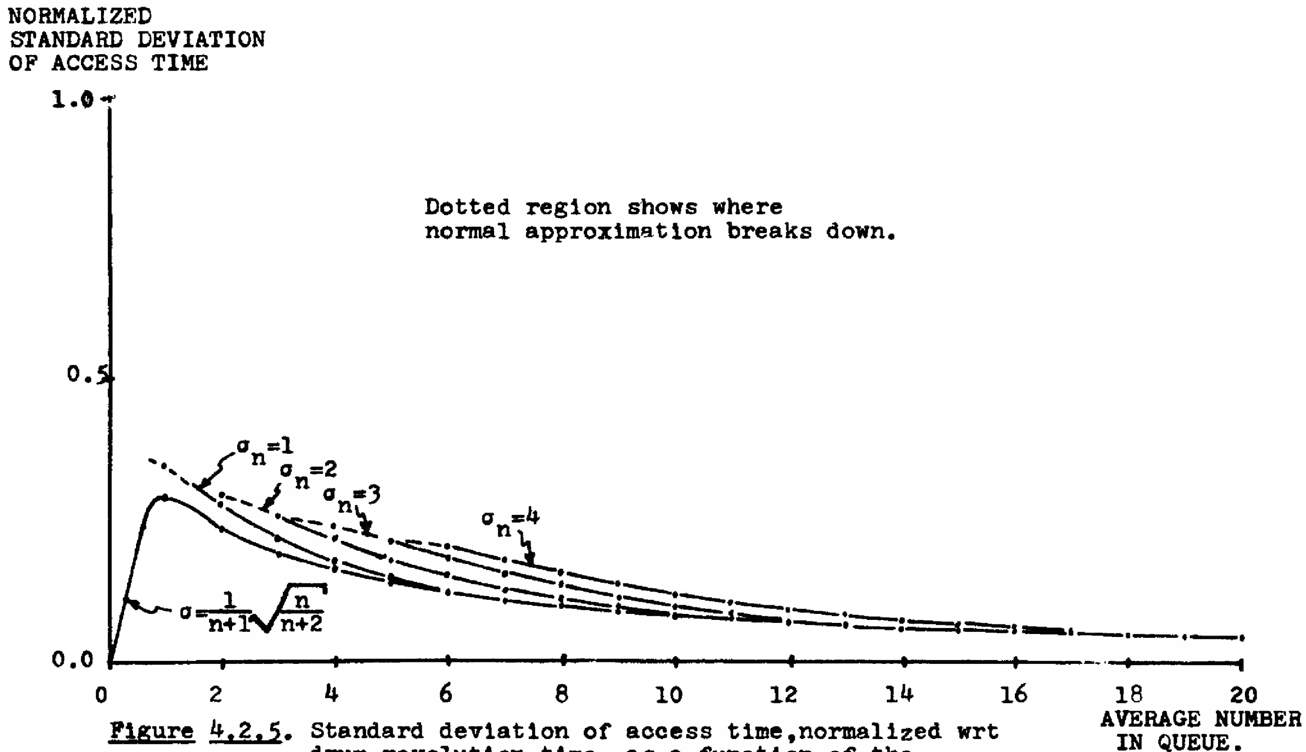


Figure 4.2.5. Standard deviation of access time, normalized wrt drum revolution time, as a function of the average number in the queue.

4.3. Examination of Shortest Access Time Queue.

The solution to a queueing problem in which the policy is based on a continuous number of priorities, such as the shortest job first and shortest access time queues, is not easily obtainable. In particular no solution has yet been obtained for a finite requesting population, under a shortest-job-first type queue discipline, since the arrival rate of requests tends to depend heavily on the size of the queue and the service time. As the queue become full, the rate of arrival of requests tends to slacken because there are fewer members of the requesting population outside of the service system. In this section we will derive a set of approximate equations for the number in the queue as a function of input and service parameters, and indicate an iterative procedure for solving them.

We suppose that the queue is in statistical equilibrium, that is, the system has been in operation sufficiently long that the time average of number in the system is constant. We shall use the following notation:

n = the mean number in the queue.

W_q = mean wait on a request in the queue.

T = drum revolution time.

T_s = mean service time.

T_t = mean transfer time.

T_a = mean access time.

a = mean arrival rate.

b = mean service rate.

A = mean interval till the next request from one process, from the time it resumes.

s = mean number of pages per segment.

m = number of sectors around the drum.

N = population size, i.e., the total number in the queue, plus the number in service, plus the number generating requests.

r = traffic intensity ratio, i.e., the average number of busy channels.

In the previous section we saw that due to independence of requests, random segment lengths, andrandom present drum position, that at each request-granting time, each request was equally likely to be next out. We have a series of Bernoulli trials, then, with a probability of $\frac{1}{k}$ of a particular request being picked at a given trial, and probability $(1 - \frac{1}{k})$ of being overlooked, where k is the number in the queue at the time of the trial. On the average we can say that the probability of being chosen on any trial is approximately $\frac{1}{n}$, where n is the average number in the queue. Therefore the probability of being chosen on the k^{th} requesting-granting time after a request enters the queue is, on the average, given by a geometric distribution, which we denote by P(k).

Then $$P(k) = (1 - \frac{1}{n})^{k-1}(\frac{1}{n}) \tag{1}$$

We wish to determine the waiting time of a request in the queue. The z-transform of equation (1), which we denote by $p_k^t(z)$, is

$$p_k^t(z) = \sum_{k=1}^{\infty} (1 - \frac{1}{n})^{k-1}(\frac{1}{n})\ z^k$$

which can be reduced to the closed form

$$p_k^t(z) = \frac{z}{n - (n - 1)z} \tag{2}$$

The mean number of trials before the given request is next out is

$$\overline{k} = \frac{d}{dz}\ p_k^t(z)]_{z=1} = n \tag{3}$$

And the variance is

$$\sigma_k^2 = [\frac{d^2}{dz^2}\ p_k^t(z) + \frac{d}{dz}\ p_k^t(z) - [\frac{d}{dz}\ p_k^t(z)]^2]_{z=1}$$

$$\sigma_k^2 = n(n - 1) \tag{4}$$

The average wait is just the average number of service intervals that must pass while a request is in the queue. If a request arrives just before a service begins it must wait only $(n - 1)$ intervals; if it arrives just after a service begins, it must wait n intervals, as given by equation (3). On the average, then, it must wait $(n - \frac{1}{2})$ service intervals. The wait in the queue is therefore

$$W_q = (n - \tfrac{1}{2})\ T_s \tag{5}$$

We suppose that each interval is of duration T_s. where T_s is the mean service time, and

$$T_s = T_a + T_t$$

and where T_a is the mean access time. For n in the queue, we can use equation (11) of Section 4.2, so that

$$T_s = \frac{T}{n+1} + T_t \qquad (6)$$

It is recalled that $T_a = T/(n+1)$ is an approximation, becoming more accurate with increasing n. The mean transfer time is the time to service the mean number of pages per segment, which is

$$T_t = T\,\frac{s}{m} \qquad (7)$$

where s = mean number of pages per segment,

m = number of sectors around the drum.

By putting (7) into (6) we obtain

$$T_s = T\left(\frac{1}{n+1} + \frac{s}{m} \right) \qquad (8)$$

We have noted that the shortest access time queue is a random output queue, so that we can use the result of Appendix 1, which says that

$$\frac{\text{Mean number in the service system}}{\text{mean service rate}} = \frac{\text{mean number in queue}}{\text{mean arrival rate}} \qquad (9)$$

where the mean service rate is b = 1/(mean service time), and the mean arrival rate is a = 1/(mean arrival time).*

*We are assuming as in Section 3.2 that arrivals are Poisson, and that segment lengths are Poisson distributed. That is, the probability of exactly k requests in a time t is

$$P(k,t) = \frac{(at)^k}{k!}\, e^{-at} \qquad t \geq 0$$

(continued)

Equation (9) is exact only when the arrival and service rates are independent of the number in the queue, which is not the case in the finite population system we are discussing. We can use equation (9) because there must exist an equivalent infinite-population system whose equilibrium arrival rate is the same as the arrival rate to the shortest access system when it is in equilibrium. We proceed to substitute the appropriate quantities into (9) and then solve for n, the mean number in the at equilibrium.

First note that r, the <u>traffic intensity ratio</u>, is

---

where a is the arrival rate at equilibrium. The probability of finding exactly k segments in a block of n pages is

$$P(k,n) = \frac{(n/s)^k}{k!} e^{-n/s} \qquad n=1,2,3,\ldots$$

where s is the mean number of pages per segment. The waiting time between poisson arrivals is

$$\begin{aligned} P(t)dt &= P(\text{no arrivals during time interval } t) \\ &\quad \times P(\text{one arrival in time interval } dt) \\ &= \left.\frac{(at)^k}{k!} e^{-at}\right]_{k=0} (a)(dt) \end{aligned}$$

so that $P(t) = ae^{-at} \qquad t \geq 0.$

also the time average number of busy channels:

$$r = \frac{a}{bc} = \frac{\text{mean arrival rate}}{\text{mean service rate}}$$

where c = number of channels,

b = mean service rate

a = mean arrival rate.

The mean number in the service system is just (n+r). Now if the interarrival time for one working process is A, then at equilibrium it must be, for all working processes,

$$\frac{A}{(N - n - r)} = \frac{1}{a} \tag{10}$$

Because (N-n-r) are not in the service system, and are therefore making requests. We can now fill in (9) to get:

$$\frac{(n+r)}{1/T_s} = \frac{n}{\frac{(N - n - r)}{A}} \tag{11}$$

We define a quantity R to be

$$R = \frac{T_s}{cA} = \frac{T}{cA}\left(\frac{1}{n+1} + \frac{s}{m}\right) \tag{12}$$

Note that R is an intensity ratio for one process, and

$$T_s = R\ A\ c \tag{13}$$

Then the intensity ratio r is

$$r = \frac{a}{bc} = \frac{T_s}{\frac{A}{(N - n - r)}} = R\ (N - n - r) \tag{14}$$

Solving (14) for r, we find

$$r = (N - n)\frac{R}{1 + R} \tag{15}$$

After putting (15) into (11) and performing the appropriate algebraic manipulation, we find

$$n = \frac{N}{1 + \frac{(1 + R)^2}{Rc(n + NR)}} \qquad (16)$$

The form of equation(16) has been chosen because it is solvable by a process known as relaxation (or iteration), in which a guess at n is put into the right side of (16), keeping in mind that $R = R(n)$. A new value of n is obtained. This new value of n is placed into the right side of (16) as before, yielding yet another value of n. This process is continued until the new value of n is the same as the previous value. It was found that (16) converges rapidly, within five cycles.

Collecting the results,

$$n = \frac{N}{1 + \frac{(1 + R)^2}{Rc(n + NR)}} \qquad (17a)$$

$$W_q = (n - \tfrac{1}{2})\ R\ A\ c \qquad (17b)$$

$$T_s = R\ A\ c \qquad (17c)$$

A simulation has been carried out to test equations (17). The value of n was found to be within 1% of the simulated values; the value of W_q was within 5%. These answers were considered satisfactory in view of the approximate nature of the derivation.

Due to the nature of this problem we are unable to

say anything about the standard deviation of our results. Simulation has shown that the standard deviation of the number in queue is less than 1.0, while the standard deviation of the waiting time was in general somewhat larger than the mean. In particular, one simulation reported a maximum wait of about ten times the mean.

As a final note we want to point out that one of the basic assumptions of this section and the previous section is that the drum position is random at each request-granting time. This means that the drum positions at successive request-granting times are independent. But this need not be the case. Suppose for instance that the transfer time, T_t, is a small fraction of the drum revolution time, T (for example, suppose the average transfer time, $T_t \approx 0.1T$). Clearly, if this is the case, the drum positions at successive request-granting times are dependent, because we can say that the probability of the drum being only 0.1T away is much greater than being, say, 0.5T away. This is obviously contradictory to the assumption of independent drum positions at successive request-granting times. Consequently we expect the access time to be below the predicted values, since the probability of finding a request wanting the present drum position is greater than if the drum position is random. If the access time were smaller than the predicted values, then both W_q and n would be smaller than predicted, T_s would be smaller, and the system operation should be more efficient. Simulation has shown that this is the case, that efficiency is increased

when segments are short. In particular, since (N-n-r) processes are working, then the fraction of processes that are working is

$$\frac{(N - n - r)}{N} = \frac{N - n}{N(1 + R)} \qquad (18)$$

If n substantially decreases, by (18) the efficiency substantially increases. The greater the efficiency, the greater the number of processors that can be kept busy. It is to be noted that when T_t is of the same order of magnitude as T, or larger, then the drum positions at successive intervals become independent, and the analysis of this section is valid.

CHAPTER 5. THE SIMULATION RESULTS. CONCLUSIONS.

5.1. Introduction.

In order to observe the operation of the model of the entire drum system, which is discussed in Chapter 3, it was decided to simulate the system. Project MAC computation facilities were used; the simulation was written in SIM, a new simulation language conceived and implemented by A.L. Scherr at Project MAC (17). SIM is an augmented version of the MAD programming language, adding several new statements to those already existing in MAD. It has the powerful advantage that the logical flow of the simulation is the same as the logical flow of the actual system. Each element of the system (see Figure 3.1), namely the processes, the queue, and the drums, is specified in the simulation as an Element (which is translated into a MAD external function by a SIM pre-compiler). The inter-elemental signals shown in Figure 3.1 are implemented in SIM by system variables, which allow a signal to be transmitted from one element to another. A main program called SIMSYS coordinates the activity of the elements.

Three simulations were run. One was a simulation of the entire drum system discussed in Chapter 3. Another was a simulation of the shortest access time queue discussed in Chapter 4. Section 5.2 discusses the drum simulation, and Section 5.3 discusses the queue simulation. A third simulation was used to develop Appendix 2, and is discussed there.

5.2. The Drum Simulation.

The three elements of the simulation were the Users' Processes, the Queue, and the Drums. These elements and the signals that were passed among them are shown in Figure 3.1. The logical flow of each element is the same as shown in the flow graphs of Figures 3.8A, 3.8B, 3.10, and 3.12, where the models of the Processes, the Queue, and the Drums are depicted.

CTSS has available a random number generator, which is useful in the simulation of the Processes to generate the probability distributions discussed in Section 3.2. The random number generator returns a number between zero and one from a uniform distribution. This can be used to get numbers from other distributions in the following manner. First the cumulative distribution of the given distribution is found, which will have probabilities varying between zero and one. The random number generator can be used to select one of these values of probability. This value is substituted into the cumulative distribution which has been solved for the random variable. In the drum simulation numbers from exponential distributions were needed. Such exponentially distributed random variables can be obtained in the following manner. Suppose we want to select a random number from the exponential distribution of interarrival times, which has

been shown to be

$$P(t) = ae^{-at} \qquad t \geq 0 \tag{1}$$

Denoting the cumulative distribution by Q(t), we have

$$Q(t) = \int_0^t ae^{-at}\, dt = 1 - e^{-at} \tag{2}$$

Solving for t,

$$t = -\frac{1}{a} \ln (1 - Q(t)) \tag{3}$$

But in Q(t) all probabilities in the interval (0,1) occur uniformly, so we can use the random number generator to select a probabilty Q(t); substitution into (3) yields the desired exponentially distributed random variable, t. Equation (3) was used in the Process Model to select waiting times til the next request, and to select the number of pages in a segment.

The following data were taken during a typical simulation, for each queue discipline:

(1) per cent process idle time;

(2) waiting time in the queue;

(3) number in the queue;

(4) service times;

(5) access times;

(6) channel idle times;

(7) number of fields used per sector on the drum.

The following set of parameters was considered typical.

Fractional drum occupancy.............	.90
Number of processes..................	20.
Mean inter-request time...............	15. msec.
Mean number pages per segment.........	10.
Read-write ratio.....................	3.
Number of drums......................	3.
Number of channels each drum..........	3.
Number of fields each drum...........	256.
Number of sectors on drum.............	64.
Number of words per page..............	64.
Drum revolution time..................	16.7 msec.

The following per cents of process idle time were found for each queue discipline:

First come first served...............	55%
Shortest job first....................	44%
Shortest Access time first............	41%

Other simulations using modified sets of parameters (for example, two drums with two channels each; or longer service times, that is, more pages per segment) showed the same result--the shortest access time queue discipline results in minimum idle time. This point has been discussed under our comparison of queues in Section 3.3.5.

Probability distributions of all data were taken. Three of them were of particular interest, and are reproduced here. These were the waiting time in queue, the number in queue, and the number of fields used per sector per drum. These are plotted in Figures 5.1, 5.2, and 5.3 for each queue discipline. The means and the maximum points are

indicated. It is notable that the mean wait for First Come First Served was 17.1 msec, while for Shortest Access Time First and Shortest Job First it was significantly less, 6.8 msec for Shortest Job First and 6.3 msec for Shortest Access Time First. Again the Shortest Access Time Queue lead to the minimum wait. It is also of significance that the shape of the waiting time in queue distritution is exponential as predicted by Appendix 2. The number in the queue (Figure 5.2) was about 8 for First Come First Served, and half that for the other two queue disciplines. The number of fields per sector per drum (Figure 5.3), is not dependent on the queue, but is dependent only on the deletion policy, which is shown in Figure 3.12. It is interesting to note that it is normally distributed, and that at desired occupancy level of 90% the maximum data point was 242 out of 256 fields used (95%). The mean was 230 fields used (90%). Tnis was for a sample of 24,500 points. We conclude that occupancy levels in excess of 90% can be maintained without overflow.

The remaining three distributions are not plotted here, but we will discuss each briefly. The service distribution was found to have approximately the same shape as the number of pages per segment distribution, but it was distorted due to the inclusion of the access time in the service time. The mean service time was found to be the sum of the mean access time and the mean transfer time, as expected.

The access time distribution was uniform for First

Come First Served, with a mean of 16.7/2 msec = 8.34 msec, as expected. For Shortest Access Time First this distribution was found to follow closely the predictions of Section 4.2. The access distribution for Shortest Job First was somewhere between the First Come First Served and Shortest Access Time distributions, as expected.

Finally the channel idle time distribution showed that there was an insignificant amount of channel idle time.

Let us mention what the maximum waits in the queue were. First Come First Served had the smallest maximum wait, as expected, and Shortest Job First had the largest. Some numbers are, for the typical parameters listed on page 68,

First Come First Served..........	62.	msec
Shortest Access Time.............	65.	msec
Shortest Job First...............	100.	msec

Note that the Shortest Access Time does not cause waits too much longer than the First Come First Served Queue. Other simulations were run, in which the Shortest Job First queue was observed to have a maximum wait of 4 sec, for parameters not too different from the ones listed on page 68.

A last point: queues in which the <u>skip limit</u>* was used have a "First Come First Served" component, and are accordingly less efficient than a Shortest Access Time queue. A skip limit of ten in a Shortest Access Time queue caused its efficiency to be only slightly greater than

*Section 3.3.4.

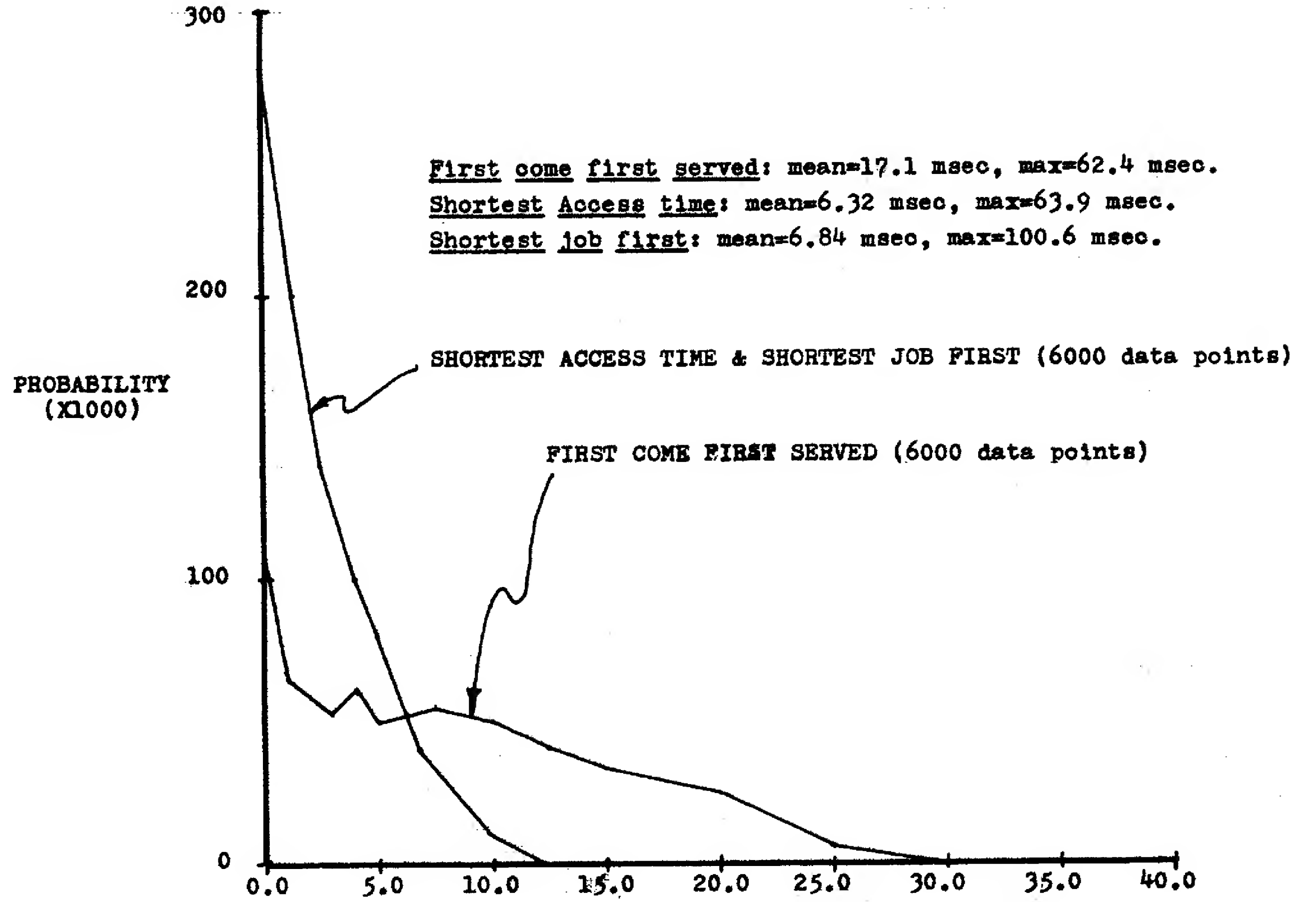


Figure 5.1. Probability densities of waiting time in queue for various queue disciplines.

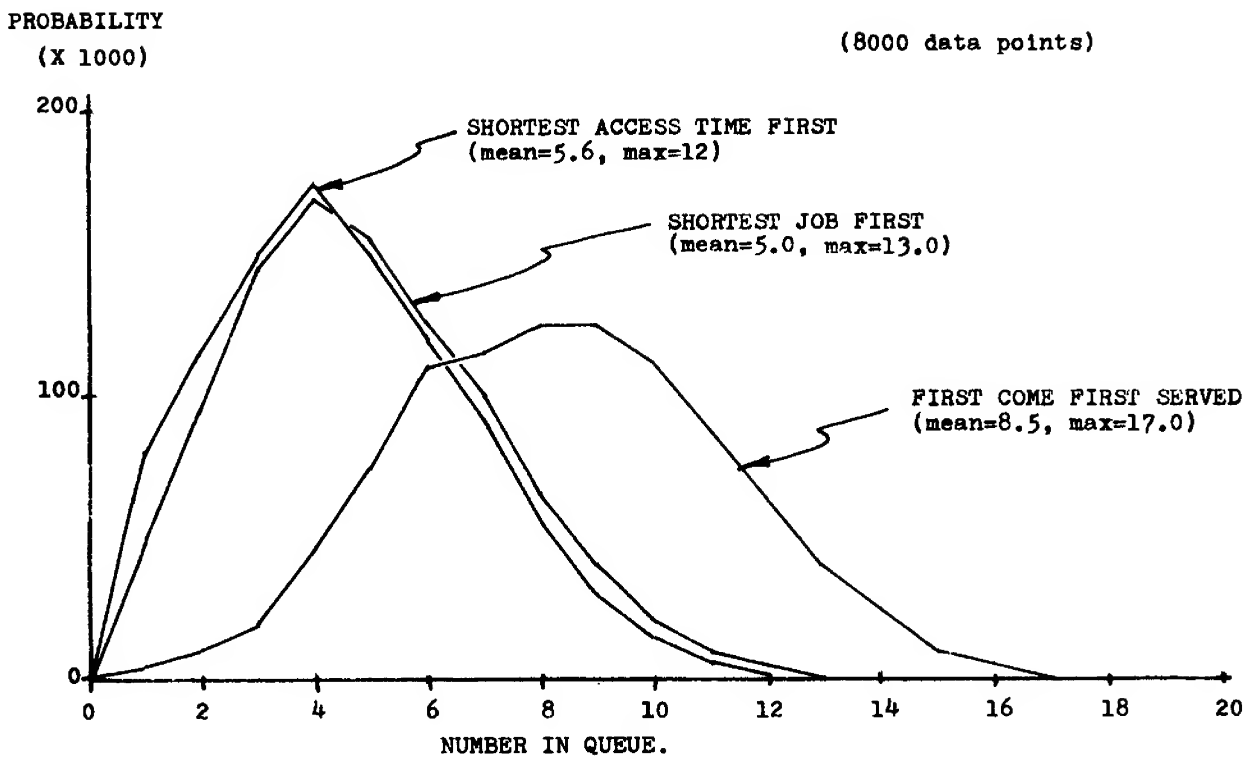


Figure 5.2. Probability densities of number in queue for various queue disciplines.

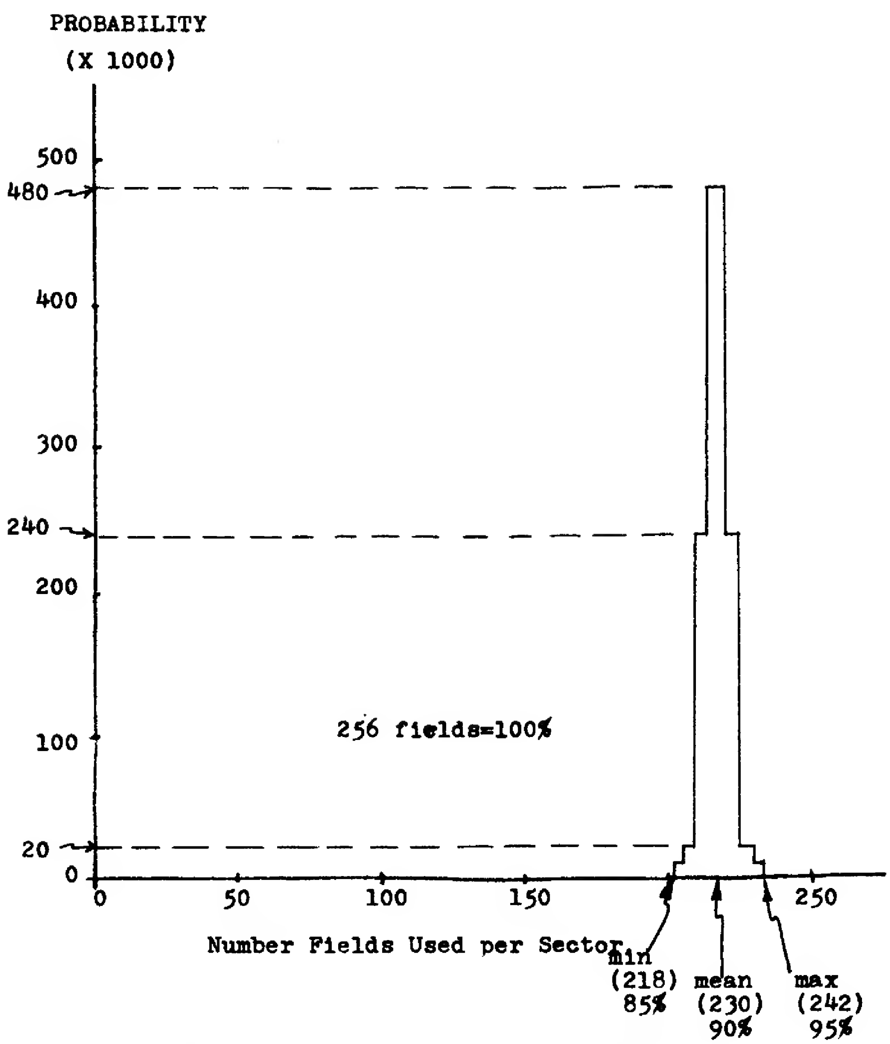


Figure 5.4. Probability density of number of fields used per sector.

a First Come First Served queue.

5.3. <u>Shortest Access Time Queue Simulation</u>.

This simulation was composed of two elements, one to make requests, and the queue. With Section 4.3, the arrival rate of requests at equilibrium is

$$a = \frac{(N - n)}{(1 + R)A} \tag{4}$$

where

$$R = \frac{T}{cA} \left(\frac{1}{n + 1} + \frac{s}{m} \right) \tag{5}$$

and T = drum revolution time,
A = inter-request times per working process,
c = number of channels,
s = mean number of pages per segment,
m = number of sectors per drum,
n = mean number in the queue.

The simulation was seeking to test equations (17) for Section 4.3, which are

$$n = \frac{N}{1 + \frac{(1 + R)^2}{Rc(n + NR)}} \tag{6}$$

$$W_q = (n - \tfrac{1}{2})\ R\ A\ c \tag{7}$$

Four single-channel (c = 1) simulations are considered here. The parameters were:

Parameter set number →	1	2	3	4
T	16.7 msec	16.7 msec	16.7 msec	16.7 msec
A	15.0 msec	5.0 msec	0.5 msec	1.0 msec
s	4.0	8.0	40.0	2.0
m	64.0	64.0	64.0	32.0
N	20.0	10.0	25.0	5.0

The results, were, for simulation samples of about 3000 data points, as follows.

Parameter set number	n predicted	n simulated	W_q predicted	W_q simulated
1	12.85	12.97	27.78 msec	26.91 msec
2	7.96	7.99	29.48 "	29.01 "
3	24.00	23.68	260.98 "	259.12 "
4	3.98	4.00	15.30 "	16.54 "

It is apparent that the agreement is good.

One last point: in Section 4.3 it was mentioned that if the drum position is not random, that is, when short segments were used, then the access times should decrease, and in particular the number in the queue and the waiting times should decrease. The following simulation verified this:

Parameters:

$T = 16.7$ msec
$A = 7.0$ msec
$s = 3.3$
$m = 64.0$

Results:	Random n	Random W_q	Function of time n	Function of time W_q
	3.86	17.11 msec	2.48	11.40 msec

There is a significant difference, and fortunately the errors are in favor of much increased operational efficiency. From Section 4.3 the efficiency is

$$\frac{N - n}{N(1 + R)}$$

Efficiency:	Random	Function of time
	$\approx 32\%$	$\approx 46\%$

5.4. <u>Conclusions</u>.

In this paper we have shown that for a segmented multiprogrammed, multiprocessor computing system, the following is true: proper maintenance of auxiliary memory can greatly improve system efficiency. We have shown how this can be done. In particular:

(1) It is generally possible to store pages consecutively on the drum, and proper deletion policy can be used to maintain occupancy levels in excess of 90%.

(2) The Shortest Access Time queue discipline is the most efficient queue for an auxiliary memory, where time is spent waiting for mechanical parts to move into some proper position.

If request sizes are large, that is if segments contain many pages, then it is not difficult to derive equations for the average number in queue, and for the average wait in the queue. If the segments are short, these equations break down, but provide an upper limit for the average number in queue and the average wait in queue: The error is in favor of increased efficiency.

(4) A reasonable probabilistic model for the processes in a segmented computing system has been given in this paper.

(5) Simulation is a particularly useful tool for analyzing problems of the complexity of computing systems, for it is frequently helpful in providing a starting point for analysis.

(6) Mixed-Policy queues may be used in drum (or disc) auxiliary memory systems when we become concerned that some requests might hive to wait inordinately long. A "skip limit" queue was found to be more efficient than a "window" queue (see Appendix 2).

5.5. Suggestions for future study.

(1) The deletion policy of the Processes model. Although it is possible to prevent drum overflow, and to maintain 90% occupancy, exactly what deletion policy is the best, if any? See chapter 2 and Section 3.4 for discussion.

(2) The "page-turning" vs. "segment-turning" allocation problem of Chapter 2 should be considered in detail.

(3) The finite population, shortest job type of queue is yet to be completely analyzed.

A P P E N D I C E S

APPENDIX 1. THE POLLACZEK-KHINTCHINE FORMULA.

In this appendix we will derive an equation which Saaty (16) refers to as the Pollaczek-Khintchine Formula (Saaty, pp 40-43). Saaty has derived it for the poisson input, single channel, equilibrium queue. We will extend the reasoning to include the c-channel server. Since we talk only of the number in the system, the queue discipline is irrelevant to our discussion, until we begin to talk of waiting times.

Suppose that arrivals occur at random according to a poisson process at a rate a per unit time, to a waiting line in statistical equilibrium, before a c-channel facility. They are served according to some arbitrary service-time distribution at a rate b per unit time per channel. We assume that if the service rate of one channel is b per unit time, then it is bc per unit time for all c channels operating together. Suppose that a departing request leaves q in the system behind, including those in service, and that some time t will elapse before the next departure. Let the waiting line increase in length by k requests during this one service interval. If the next departing request leaves q' behind in the system, we can relate q and q' as follows:

$$q' = \max(q - 1, 0) + k = q - 1 + d + k \quad (1)$$

where

$$d(q) = 0 \text{ if } q > 0$$
$$d(q) = 1 \text{ if } q = 0$$

By introducing d(q) we eliminate the max expression.

We assume that equilibrium values for the first and second moments $E(q)$ and $E(q^2)$ of the number in the system exist, where we are treating q as a random variable. We note that $E(q) = E(q')$ and $E(q^2) = E(q'^2)$ since both q and q' are assumed to have the same equilibrium distribution. We observe that since equilibrium, each departing request must leave behind identical time-independent queues, each having the same probability distribution. Now, from the definition, $d^2 = d$, and $q(1 - d) = q$. Thus, taking the expected value of (1) we have

$$E(q') = E(q) - E(1) + E(d) + E(k) \qquad (2)$$

but since $E(q) = E(q')$ we have

$$E(d) = 1 - E(k) \qquad (3)$$

During an inter-departure interval of length t we have

$$E(k) = \sum_{k=0}^{\infty} k\frac{(at)^k}{k!} e^{-at} = at \qquad (4)$$

$$E(k^2) = \sum_{k=0}^{\infty} k^2 \frac{(at)^k}{k!} e^{-at} = (at)^2 + at \qquad (5)$$

Let us denote the combined service distribution for all c channels operating in parallel by S(t). Taking the expectation of E(k) with respect to this service time distribution we see that

$$E(k) = \int_0^{\infty} (at)\ S(t)\ dt$$

$$= a\int_0^{\infty} t\ S(t)\ dt \qquad (6)$$

$$E(k) = \frac{a}{bc} = r$$

since the mean of $S(t)$ is $1/bc$. But since se have not specified $S(t)$, $E(k) = r$ is unaffected by the type of service distribution. Then

$$E(d) = 1 - r \tag{7}$$

Now, if the probability of the queue increasing by k is independent of the length of queue, q, and of d, which depends only in q, any expectation over products of r, q, and d is just the product of the respective expected values. Therefore

$$E(k^2) = \int_0^\infty ((at)^2 + at)\, S(t)\, dt$$

which is an average over all time. But

$$E(k^2) = \int_0^\infty (at)^2\, S(t)\, dt + \int_0^\infty (at)\, S(t)\, dt$$

$$= a^2\, \overline{t_s^2} + a\bar{t}_s$$

But the variance of $S(t)$, σ_s^2, is

$$\sigma_s^2 = \overline{t_s^2} - \bar{t}_s^2$$

Therefore

$$E(k^2) = a^2(\sigma_s^2 + \bar{t}_s^2) + a\bar{t}_s$$

Finally,

$$E(k^2) = a^2\sigma_s^2 + r^2 + r \qquad r = \frac{a}{bc} \tag{8}$$

If we square both sides of equation (1):

$$q'^2 = (q - 1)^2 + 2(q - 1)(d + k) + (d + k)^2$$

$$q'^2 = q^2 - 2q(1 - k) + (k - 1)^2 + d(2k - 1) \tag{9}$$

Equation (9) was obtained by using $qd = 0$, and $d^2 = d$.

Because of equilibrium,

$$E(q^2) - E(q'^2) = 0 = 2E(q)E(k - 1) + E((k - 1)^2) + E(d)E(2k - 1) \qquad (10)$$

Recall that the validity of (10) depends on the independence of q and k. Solving for E(q) and using equations (6), (7), and (8), we have the Pollaczek-Khintchine Formula:

$$E(q) = \frac{E((k - 1)^2) + E(d)E(2k - 1)}{2E(1 - k)}$$

$$= \frac{a^2\sigma_s^2 + r^2 + r - 2r + 1 + (1 - r)r - (1 - r)}{2 - 2r}$$

$$E(q) = r + \frac{r^2 + a^2\sigma_s^2}{2(1 - r)} \qquad r = \frac{a}{bc} \qquad (11)$$

Thus, once we know the variance of the service time distribution, the average number in the system, E(q) is determined. It is important to note that E(q) is an average taken over instants just following departures, and is not the time average. If $E_t(q)$ is the time average, all we can say without further argument is that

$$E(q) < E_t(q) < E(q) + 1$$

In general the average number in the service system equals the sum of the average number of busy channels (here it is $r = \frac{a}{bc}$) plus the average number in line.

To obtain the average wait in the waiting line, which we will denote by E(w), we observe that $a(E(w) + \frac{1}{bc})$ is the

expected number of arrivals during the expected time of one request in the service system, if the queue discipline is first come first served, because $\frac{1}{bc}$ is the mean service time. But this must be just the number in the system immediately after a customer departs, that is, E(q), so

$$aE(w) + \frac{a}{bc} = r + \frac{r^2 + a^2\sigma_s^2}{2(1 - r)}$$

but $r = \frac{a}{bc}$, so

$$W_q = E(w) = \frac{r^2 + a^2\sigma_s^2}{2a(1 - r)} \qquad (12)$$

We have pointed out that r is just the number of busy channels and that E(q) is the expected number in the system. Inspection of (11) will show that the number in line, L_q, must be

$$L_q = \frac{r^2 + a^2\sigma_s^2}{2(1 - r)}$$

We have the interesting and important result

$$W_q = \frac{L_q}{a} \qquad (13)$$

Notice that this is exact only if the number in the system, E(q), is independent of the service time or the arrival rate, as pointed out after equation (10). We also note that if $(W_q + \frac{1}{bc})$ is the time of one customer in the service system, then $bc(W_q + \frac{1}{bc})$ is one more than the number in the system, E(q). This is so because if E(q) are in the system, then E(q) - 1 service intervals pass while one request is in the system. Therefore bcW_q = E(q) and we have the

second result

$$W_q = \frac{E(q)}{bc} \tag{14}$$

In words:

$$W_q = \frac{\text{average number in the line}}{\text{average arrival rate}}$$

$$= \frac{\text{average number in the system}}{\text{average service rate}}$$

These are true for arbitrary service distributions.

It is interesting to note that if the service times are exponential, that is, the service follows a poisson law, then the interval between departures is given by

$$S(t) = bce^{-bct} \qquad t \geq 0$$

It is a well-know fact that for this type of distribution the variance equals the mean squared, that is,

$$\sigma_t^2 = \int_0^\infty t^2 bce^{-bct}\, dt - [\int_0^\infty tbc\, e^{-bct}\, dt]^2$$

$$= (1/bc)^2$$

Substitution of this into (11) yields

$$L_q + r = r + \frac{r^2}{1 - r} \tag{15}$$

From which it follows that

$$L_q = \frac{r^2}{1 - r} \tag{16}$$

and

$$E(q) = \frac{r}{1 - r} \tag{17}$$

Consider for a moment the geometric distribution

$$P(k) = r^k(1 - r) \tag{18}$$

It is known that this distribution describes the number in a service system with exponential input and output (Saaty, 17, pp 38ff). The expected number in the system is

$$L = \sum_{k=0}^{\infty} k\, r^k (1 - r)$$

$$= (1 - r) r \frac{d}{dr} \sum_{k=1}^{\infty} r^k$$

$$= \frac{r}{1 - r}$$

which is the same as (17). Then we can find the variance of (18) which is

$$\sum_{k=0}^{\infty} k^2\, r^k (1 - r) - L^2 = (1 - r) r \frac{d}{dr}\, r \frac{d}{dr} \sum_{k=0}^{\infty} r^k - L^2$$

$$\sigma_L^2 = \frac{r}{(1 - r)} + \frac{r^2}{(1 - r)^2}$$

$$\sigma_L^2 = L(L + 1) \qquad (19)$$

We have the result that for the exponential input, exponential output system the number in the system is given by (17), the number in queue by (16), and the variance of the number in the system by (19). The results of this section will hold for queues in which the discipline is random as well as for first come first served. They hold for random disciplines because, on the average, the number of service intervals that must pass before service is the same as for first come first served. This is seen in Section 4.3. In fact the equation for the mean number in the queue, L_q is accurate if the following conditions are satisfied:

(1) all requests stay in the queue until served;

(2) the service time distribution for all channels is the same, with parameter b;

(3) channels serve one at a time;

(4) a channel serves the next request, if any are are waiting in the queue, as soon as it finishes with the last request.

A little thought will show that if these four rules hold, the length of the queue is the same for all disciplines, although the mean wait, W_q, will vary. (Morse, 13, p. 117).

APPENDIX 2. WAITING TIME IN A SHORTEST ACCESS QUEUE.

In this appendix the probability density function for the waiting time in a shortest acess time queue is derived. We define the following random variables:

A = r.v. of access time, taking on values a.

N = r.v. of number of requests to exit the queue before a given request exits, taking on values n.

P = r.v. of number of pages per segment, taking on values p.

R = r.v. of number of requests in the queue, taking on values r.

T = r.v. of transfer time, taking on values t.

W = r.v. of waiting time in queue, taking on values w.

Since at each trial (request-granting time) all requests are assumed to be equally likely to exit next (Section 4.2) the distribution of N is geometric. As on page 58, equation (1), the conditional distribution of N given that R are in the queue is

$$P_{N/R}(n/r) = (1 - \frac{1}{r})^{n-1}(\frac{1}{r}) \qquad (1)$$

where R is the random variable of the number of requests in the queue. Denoting the density function of R as $P_R(r)$:

$$P_{NR}(n,r) = P_{N/R}(n/r)\ P_R(r) \qquad (2)$$

We are interested in the wait in queue, so we have defined the random variable of wait to be W. Then

$$P_{WNR}(w,n,r) = P_{W/NR}(w/n,r)\ P_{N/R}(n/r)\ P_R(r) \qquad (3)$$

For a single channel queue the wait in the queue is N access times plus N transfer times. As in Section 3.2 we assume the number of pages per segment to be a random variable, P, where

$$P_P(p) = c^2 p\ e^{-cp} \tag{4}$$

and c is a constant proportional to the mean number of pages per segment. If T' is the drum revolution time and S the number of sectors around the drum, then the transfer time for one pages is T'/S. Denoting the random variable of transfer time by T, we have for the density function of T:

$$P_T(t) = k^2 t\ e^{-kt} \tag{5a}$$

with the constant k defined as

$$k = \frac{S}{\overline{N}_s T'} \tag{5b}$$

and $\overline{N}_s$ is the average number of pages per segment. The wait in the queue is, from above

$$W = N\ (A + T) = NA + NT = y + z$$

with y = NA and z = NT.

From Section 4.2, the cumulative distribution of the access time is

$$P[A \leq a_o] = 1 - (1 - a_o)^N$$

But y = NA. Then

$$P[y \leq a] = P[A \leq \tfrac{a}{N}] = 1 - (1 - \tfrac{a}{N})^N$$

$$= 1 - [(1 - \tfrac{a}{N})^{-N/a}]^{-a} \tag{6}$$

Now let $u = -a/N$. Then

$$P[y \leq a] = 1 - [(1 + u)^{1/u}]^{-a} \quad (7)$$

For large N, u approaches zero and we know

$$\lim_{u \to 0} [1 + u]^{1/u} = e \quad (8)$$

Thus for large N,*

$$P[y \leq a] \approx 1 - e^{-a}$$

and the density function for the access time component of the wait in queue is

$$P_y(a) = \frac{d}{da} P[y \leq a] \approx e^{-a} \quad (9)$$

Using an elementary probability transformation, the density function for $z = NT$ is

$$P_z(b) = \frac{1}{N} P_T\left(\frac{b}{N} \right) = \frac{k^2}{N^2} b\, e^{-(kb)/N}$$

Defining $K_n = k/N = S/T'N\overline{N}_s$ we have

$$P_z(b) = K_n^2\, b\, e^{-K_n b} \quad (10)$$

Since A and T are independent random variables, the conditional density function for W, given N and R is

$$P_{W/NR}(w) = \int_{-\infty}^{\infty} P_y(w-x)\, P_z(x)\, dx$$

the convolution of $P_y(a)$ and $P_z(b)$. This evaluates to be

$$P_{W/NR}(w) = \frac{K_n^2}{(K_n - 1)^2}\, e^{-w} \quad (12)$$

*This approximation is surprisingly good for N > 10.

Recalling equation (3),

$$P_{WNR}(w,n,r) = P_{W/NR}(w/n,r)\ P_{N/R}(n/r)\ P_R(r)$$

Putting (1) and (12) into (3),

$$P_{WNR}(w,n,r) = \left[\frac{K_n^2}{(K_n-1)^2} e^{-w} \right]\left[(1-\tfrac{1}{r})^{n-1}(\tfrac{1}{r})\right]\left[P_R(r)\right]$$

$$P_{WNR}(w,n,r) = \frac{(k/n)^2}{((k/n)-1)^2} e^{-w} (1-\tfrac{1}{r})^{n-1}(\tfrac{1}{r})\ P_R(r)$$

If N is large, as it is assumed to be, then $P_R(r)$ is approximately Normal by the Central Limit Theorem, and

$$P_{WNR}(w,n,r) \approx \frac{(k/n)^2}{((k/n)-1)^2} e^{-w} (1-\tfrac{1}{r})^{n-1}(\tfrac{1}{r}) \frac{1}{\sqrt{2\pi}\sigma_r} e^{-(r-\bar{R})^2/2\sigma_r^2} \tag{13}$$

which is the required joint density function of waiting time in the queue. The simulation has shown that for the mean queue length, $\bar{R}$, greater than about 10 with $\sigma_r \ll \bar{R}$ (which is the case when $\bar{R} > 10$) this approximation is valid. Thus in the steady state situation, it is clear that the probability density for waiting time in the queue is approximately exponential, a fact verified by simulation (Section 5.2).

APPENDIX 3. DESCRIPTION OF A MIXED-POLICY QUEUE.

The queue described in this section has been proposed as a shortest access time queue, but one for which we are concerned that a particular request may be continually overlooked due to the random nature of selection. Consider for example a queue which has many requests in it (at least thirty). Such a queue might occur if it were decided to request pages singly instead of in segments. In Section 4.3 it is shown that the waiting time of a request until it leaves a random output queue is given by a geometric distribution, with the expected wait equal to n service intervals, where n is the average number in the queue. Now if n is large, then it is conceivable that a request might have to wait for a very long time: the variance of the geometric distribution is $(n)(N - 1) \approx n^2$ if n is large.

Consider the queue shown in Figure A3.1. A new request is always added to the bottom of the stack. A section of the stack, of length N, is considered, the remaining requests in the queue being ignored for the while. We shall refer to the portion of the queue under consideration as being viewed through a window, of size N. The top of the window is always at the top of the stack. The requests in the window are labelled $R_1, R_2, \ldots, R_N$, and are considered according to the shortest access time first queue discipline. Whenever the request marked R_1 is removed, the window is moved down until its top coincides with the next request R_1. It

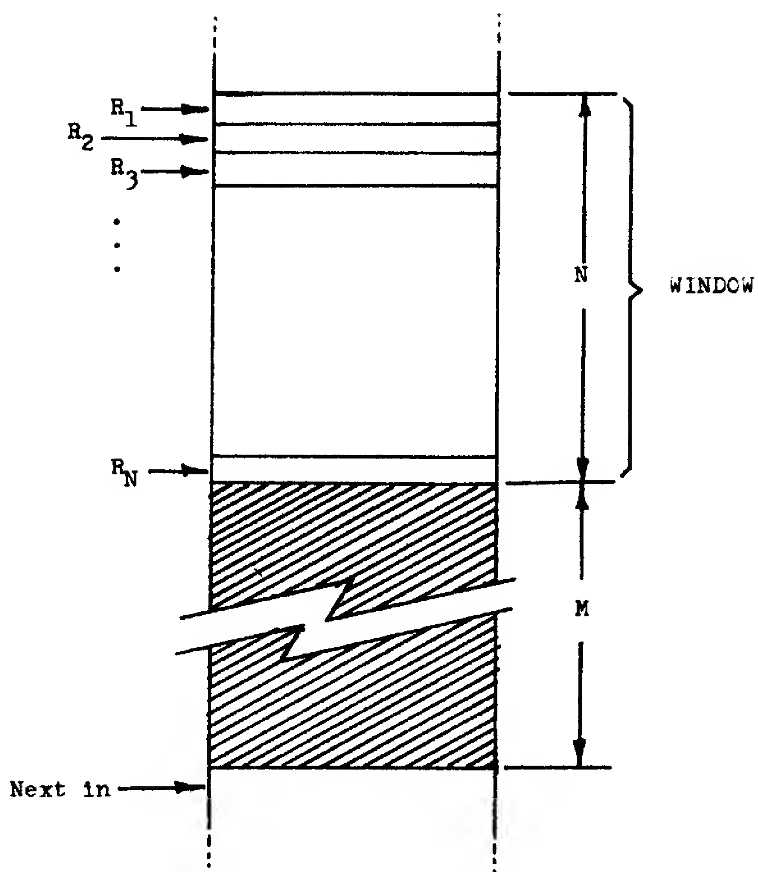


Figure A3.1. Structure of a Mixed-Policy Queue.

appears that R_1 might have to wait until the N^{th} service time before it leaves, but no longer (by then it would be the only request in the window); thus it seems that an upper bound can be placed on the waiting time in the window, namely (N - 1) service intervals. But this is not so. Consider the request marked R_N. Suppose by some quirk of fate that requests are serviced as follows. $R_1, R_2, \ldots,$ $R_{N-1}, R_{N+1}, \ldots, R_{2N}, R_N$. This would happen if $R_1, \ldots, R_{N-1}$ were serviced before R_N; but then the window has become positioned at R_N, and the next (N - 1) requests could be serviced before R_N. It is clear that the maximum wait in the window is 2(N - 1). Since the arrival rate is given by an average, the expected wait before reaching the window is M; an upper bound to the wait is M + 2(N - 1) service intervals. We are assuming $M > N$.

To find a lower bound on the waiting time, consider the following argument. Suppose a request enters the queue just before the window makes a jump of N, then suppose the window moves one position at the end of each service interval. The request in question would then wait only (M - N) service intervals to reach the window. Then suppose it were let out immediately. The minimum wait is therefore M - (N-1) = (M - N + 1). We have set an upper limit on the waiting time:

$$W_{max} = (M + 2(N - 1))\,\overline{t}_s \qquad (1)$$

and the lower limit of waiting time is

$$W_{min} = (M - (N - 1))\ \bar{t}_s \qquad (2)$$

Equations (1) and (2) assume that $M \gg N$.

On the average the window is not full. We can think of the problem as a flow problem, with requests flowing into the bottom of the wondow at the rate of one per service interval, and filtering out through the window at the same rate. Let us imagine one of the requests being tagged so that we can keep track of it. If we know on the average how far down the window a request moves before it exits then we know the mean wait in the window. Simulation of the problem for several window sizes was carried out, and it was found that on the average the tagged request went half way down the window before exiting. Then we can write

$$W_{av} = (M + \frac{N}{2})\ \bar{t}_s \qquad (3)$$

Figure A3.2 shows the probability densities of a request being at various positions in the window. It is to be noticed that the tagged request spends considerably more time at the upper and lower ends of the window then at the center. The standard deviation was found to be 0.8 of the mean, so the contention that the request is a $\frac{N}{2}$ on the average is not too certain. This implies that the probabilities of W_{max} and W_{min} are not small. Figure A3.3 shows the probability density of window jumps. The average window jump is about $\frac{N}{3}$. Figure A3.4 shows the following: the mean position reached

by the tagged request, and the mean window movement when it moves, both as a function of window size.

Recall that for efficient access time queueing the mean in the queue had to be at least eight. Hence we would require that the window length be $N \geq 16$. But since $M > N$, the overall queue would have to have an expected length $M + N \geq 32$.

It appears that the use of the minimum access time queue without the window, but with the "skip limit" mentioned in Section 3.3.5 is better for the following reason. The skip limit could be set to an upper limit of $2(N - 1)$ so that the maximum wait for that queue would be the same as given by equation (1), but with $M = 0$. Since the "skip limit" queue with the same maximum is longer than the corresponding "window" queue, the access time is shorter, and more efficient queueing is had.

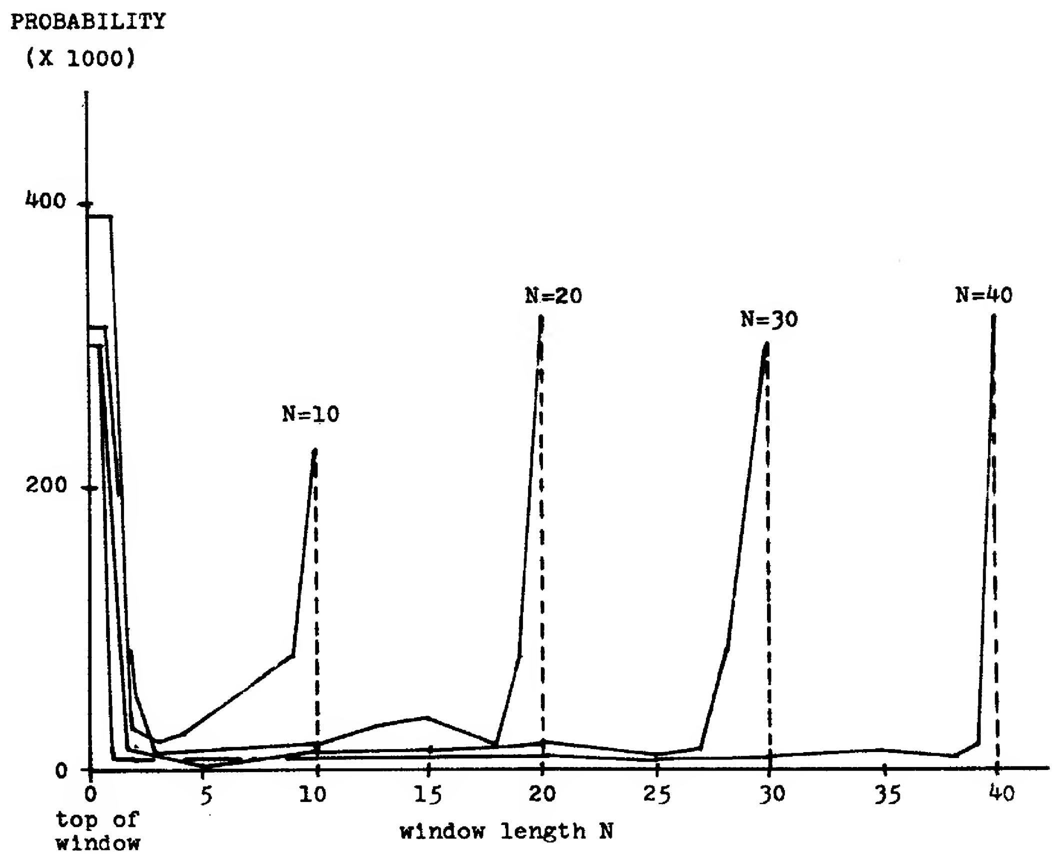


Figure A3.2. Probability density of position of tagged request.

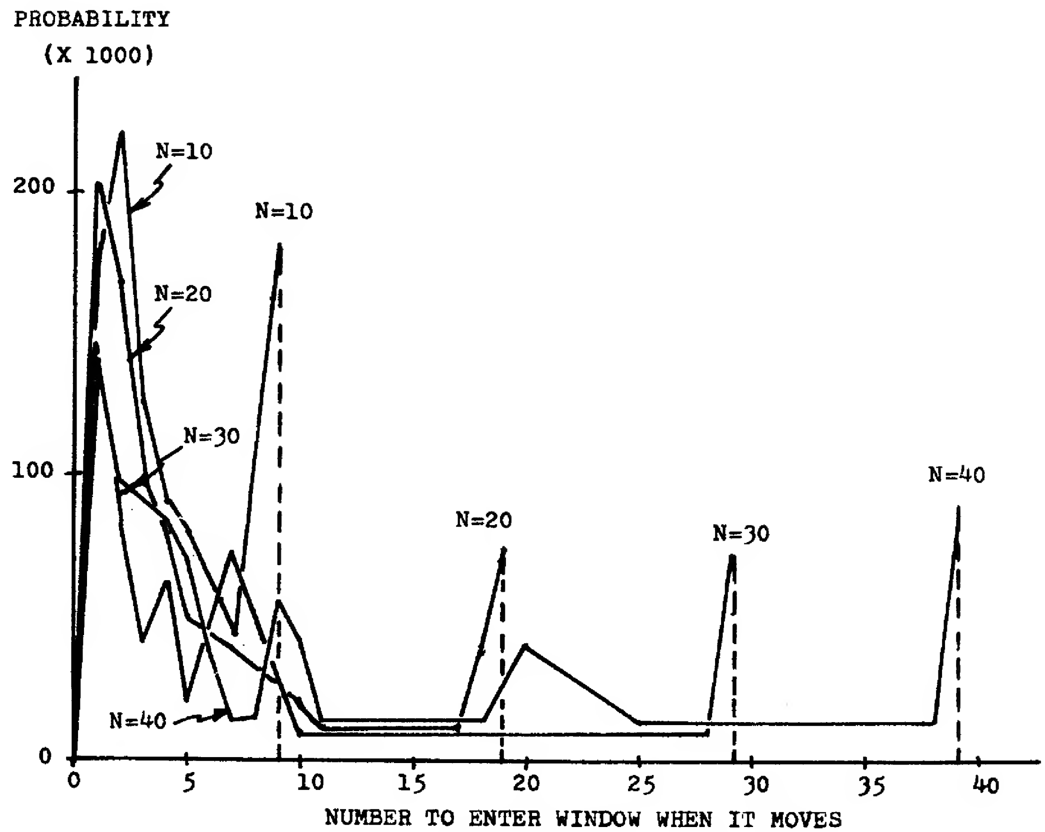
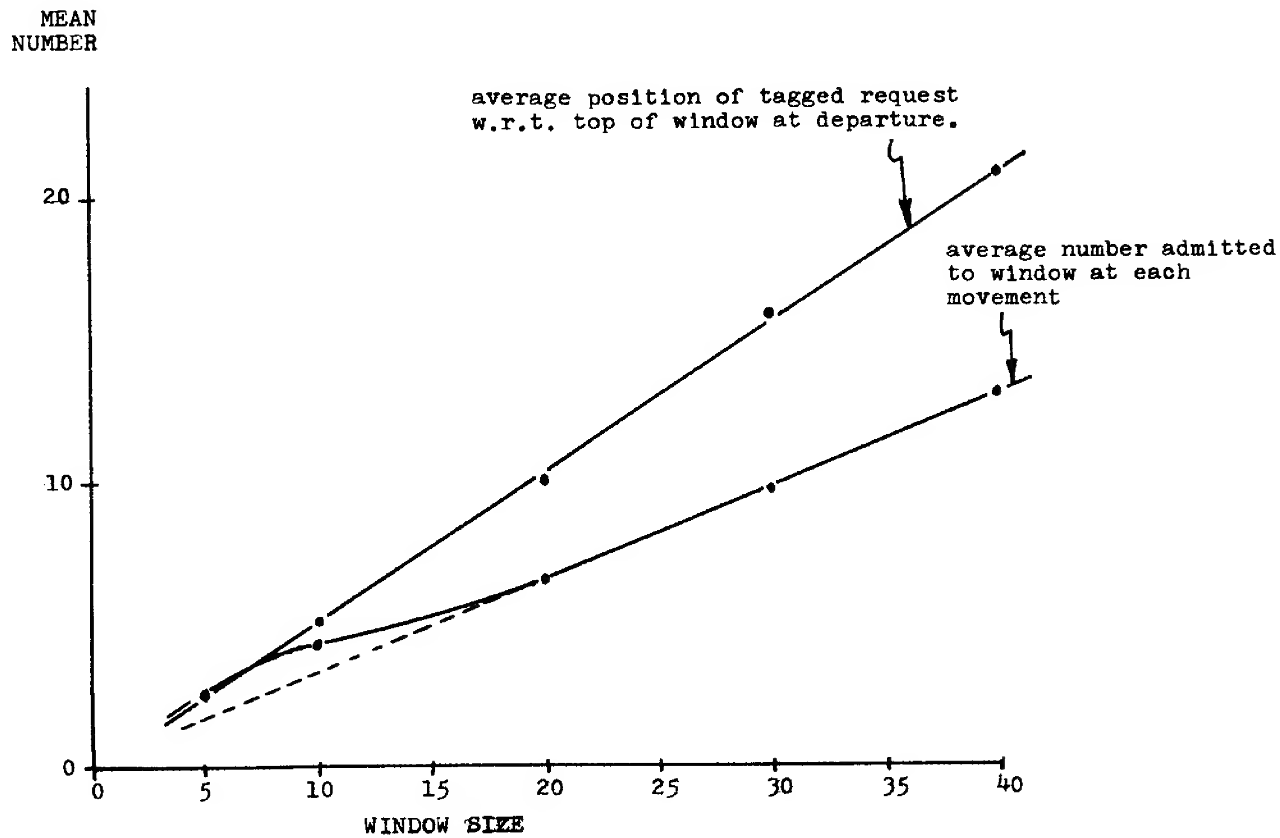


Figure A3.3. Probability density of number entering window when it moves.



<u>Figure</u> <u>A3.4</u>. A comparison of window movement and average time spent in the window.

APPENDIX 4. A CONTINUOUS-TIME MARKOV MODEL.

With Howard (11, pp. 92ff) we define a rate Matrix $[A]$, having elements a_{ij}. The rate matrix is similar to the familiar Markov transition probability matrix except that the elements a_{ij} represent transition rates from the i^{th} to the j^{th} state. The rates are assumed to be taken from exponential distributions. A transition matrix, then, is a discrete form of a rate matrix. Since we consider an equilibrium system the overall rate of change must be zero. Define a state probability vector P, where $P = [p_1, p_2, \ldots, p_M]$ and p_i is the probability that the system has i requests in it. Because of equilibrium,

$$[P][A] = 0 \qquad (1)$$

We make the following assumptions.

(1) All requests join the queue and do not leave until service is complete.

(2) Each channel serves one request at a time, and does not begin the next request until the present request is finished.

(3) As soon as a channel becomes idle, the next request enters service, provided there are some in the queue.

(4) The queue discipline is first come first served, or else random. For any other queue discipline that satisfies (1) through (3) the expressions for state probabilities and average number in line are the same, but the waiting time in the queue is not the same. See closing remarks of Appendix 1.

We use the following notation:

M = the size of the finite number number in the total population being considered--it is the sum of the number in the service system plus the number making requests.

a = mean request rate per requestor, where $1/a$ = mean interarrival interval per requestor.

b = mean service rate per channel, where $1/b = \bar{t}_s$, the mean service time.

c = the number of parallel channels providing service.

p_n = the probability of the service system having n of the M possible requestors in it.

If the system in in state n (indicating that n requests are in the service system, and that (M - n) are remaining outside in the requesting population) then the rate of exit to the state (n+1) is nb for $n \leq a$ and is cb for $n > o$. We have the rate matrix

$$[A] = \begin{bmatrix} -Ma & Ma & 0 & 0 & \cdots & \cdots \\ b & -b-(M-1)a & (M-1)a & 0 & \cdots & \cdots \\ 0 & 2b & -2b-(M-2)a & (M-2)a & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & cb & -cb-2a & 2a & 0 \\ \cdots & \cdots & 0 & cb & -cb-a & a \\ \cdots & \cdots & 0 & 0 & cb & -cb \end{bmatrix}$$

At the c^{th} row the matrix is

$$\begin{bmatrix} \cdots & (c-1)b & -(c-1)b-(M-c+1)a & (M-c+1)a & 0 & \cdots \\ \cdots & 0 & cb & -cb-(M-c)a & (M-c)a & \cdots \\ \cdots & 0 & 0 & cb & -cb-(M-c-1)a & \cdots \end{bmatrix}$$

Because of equation (1) we can write

$$-Map_o + bp_1 = 0$$

$$Map_o - bp_1 - (M-1)ap_1 + 2bp_2 = 0$$

and in general

$$(M-n+1)ap_{n-1} - nbp_n - (M-n)ap_n + (n+1)bp_{n+1} = 0 \qquad 1 \leq n \leq c$$

$$(M-n+1)ap_{n-1} - cbp_n - (M-n)ap_n + cbp_{n+1} = 0 \qquad c \leq n \leq M$$

Adding the n^{th} and the $(n-1)^{th}$ equations, which is equivalent to adding adjacent columns in [A], we have by recursion

$$p_1 = Mrp_o$$

$$p_2 = \frac{M-1}{2} rp_1 = \frac{M(M-1)}{2} r^2 p_o$$

$$\vdots$$

$$p_n = \frac{M(M-1)(M-2)\ldots(M-n)}{n!} r^n p_o$$

so that

$$p_n = \frac{M!}{n!(M-n)!} r^n p_o \qquad 0 \leq n < c$$
$$p_n = \frac{M!}{c!(M-n)!} r^n \frac{1}{c^{n-c}} p_o \qquad c \leq n \leq M \tag{2}$$

where $r = \frac{a}{b}$. p_o is found from the requirement that

$$\sum_{n=0}^{M} p_n = 1 \tag{3}$$

$$p_o = \frac{1}{\sum\limits_{n=0}^{c-1} \frac{M!}{n!(M-n)!} r^n + \sum\limits_{n=c}^{M} \frac{M!}{c!(M-n)!} r^n \frac{1}{c^{n-c}}} \tag{4}$$

If h is the average number of processes actually in operation, k the number being serviced, and L_q the number in line, then

$$k + h + L_q = M \tag{5}$$

and because of equilibrium

$$\frac{h}{k} = \frac{a}{b} = r \tag{6}$$

The number being serviced is

$$k = \sum_{n=0}^{c-1} np_n + c \sum_{n=c}^{M} p_n = c - \sum_{n=0}^{c-1} (c-n)\ p_n \tag{7}$$

The number in line is

$$L_q = \sum_{n=0}^{M} (n-c)\ p_n \tag{8}$$

and as usual the waiting time in the line is

$$W_q = \frac{L_q}{a} \tag{9}$$

The number in the system is

$$L = L_q + k = \sum_{n=0}^{M} n\ p_n \tag{10}$$

The efficiency is

$$\frac{\text{number of working processes}}{M} = \frac{M - L_q - k}{M}$$

$$= \frac{M - \sum_{n=0}^{M} n\ p_n}{M}$$

The summations can be evaluated on a computer without too much difficulty if the factorials are expressed as logarithms, and use is made of the fact that

$$n! = \exp\left[\sum_{i=1}^{n} \ln (i)\right]$$

It is interesting that a closed form for (10) can be obtained when there is one channel, i.e., when c=1. In this case equations (2) become

$$p_n = \frac{M!}{(M-n)!} r^n p_o \qquad n=0,1,\dots,M \tag{11}$$

and (4) becomes

$$p_o = \frac{1}{\sum_{n=0}^{M} \frac{M!}{(M-n)!} r^n} \qquad r = \frac{a}{b} \tag{12}$$

Then L is the number in the system, and $L-r=L_q$ is the number in the line.

$$L = p_o \sum_{n=0}^{M} \frac{nM!}{(M-n)!} r^n \tag{13}$$

Consider

$$\frac{M-L}{p_o} = \sum_{n=0}^{M} \frac{(M-n)M!}{(M-n)!} r^n = \sum_{n=0}^{M} \frac{M!}{(M-n-1)!} r^n \tag{14}$$

expanding (14) we find

$$\frac{M-L}{p_o} = M + M(M-1)r + M(M-1)(M-2)r^2 + \dots \tag{15}$$

But

$$\frac{1}{p_o} = 1 + Mr + M(M-1)r^2 + M(M-1)(M-2)r^3 + \dots \tag{16}$$

Comparison of (15) and (16) reveals that

$$\frac{M-L}{p_o} = \left(\frac{1}{p_o} - 1 \right) \frac{1}{r} \tag{17}$$

Solving for L,

$$L = M - \frac{1 - p_o}{r} \tag{18}$$

All that is needed to find L is an evaluation of p_o, not an evaluation of each p_n as well. The number in the queue is

$$L_q = L - r = M - r - \frac{1 - p_o}{r} \qquad (19)$$

So that the waiting time in queue is

$$W_q = \frac{L_q}{a} = \frac{M}{a} - \frac{r}{a} - \frac{1 - p_o}{ar}$$

$$W_q = \frac{1}{a} \left(M - \frac{1 - p_o}{r} \right) - \bar{t}_s \qquad (20)$$

where $\bar{t}_s = \frac{1}{b}$, the mean service time.

The interested reader is referred to A.L. Scherr's Doctoral Thesis, in which it is shown that Multiprocessor time-shared computing systems are in general, <u>accurately</u> described by Markov Models.

BIBLIOGRAPHY

1. Churchman, C.W., et al. Introduction to Operations Research (Chapter 14, "Queueing Models"). New York, John Wiley, 1957.

2. Corbato, F.J., et al. The Compatible Time-Sharing System: A Programmer's Guide. Cambridge, M.I.T. Press, 1963. Also 2nd Edition, 1965.

3. Dennis, J.B. Program Structure in a Multi-Access Computer. Cambridge, M.I.T., Project MAC Memo MAC-TR-11.

4. Dennis, J.B. An Example of Intersphere Communication And Asynchronous Parallel Processing. Cambridge, M.I.T., Project MAC Memo MAC-M-189: September, 1964.

5. Dennis, J.B. Automatic Scheduling of Priority Processes. Cambridge, M.I.T. Project MAC Memo MAC-M-187: October, 1964.

6. Dennis, J.B. Segmentation and the Design of Multi-Programmed Computer Systems. Cambridge, M.I.T. Project MAC memo: January, 1965.

7. Dennis, J.B., and Van Horn, E.c. Nesting and Recursion of Procedures in a segmented Memory. Cambridge, M.I.T. Project MAC memo.

8. Feller, W. Probability Theory and its Applications. New York: John Wiley, 1950.

9. Flores, Ivan. Derivation of a Waiting-time Factor for a Multiple-bank Memory. Journal of the American Association for Computing Machinery, Vol. II, No. 3 (July, 1964), pp 265-282.

10. Heller, Nelson B. Stochastic Models of a Multiple Access Computer. M.I.T. March 1965.

11. Howard, R.A. Dynamic Programming and Markov Processes. Cambridge: M.I.T. Press, 1960.

12. Lee, Y.W. Statistical Theory of Communication. (chapters 3-6). New York: John Wiley, 1964.

13. Morse, P.M. Queues, Inventories, and Maintenance. New York: John Wiley, 1963.

14. Patel, Nitin R. A Mathematical Analysis of Computer Time Sharing Systems. Cambridge: M.I.T. Operations Research Center Interim Technical Report No. 20, July, 1964.

15. Riordan, J. Stochastic Service Systems. New York: John Wiley, 1962.

16. Saaty, T.L. Elements of Queueing Theory. New York: McGraw-Hill Book Company, 1961.

17. Scherr, A.L. An Analysis of Time-Shared Computer Systems. Cambridge: M.I.T., Course 6 Ph.D. Thesis, June, 1965. Published as Project MAC Technical Report MAC-TR-18, August, 1965.

18. Witsenhausen, H. A Note on Asynchronous Parallel Processing. Cambridge: M.I.T. Project MAC Memo MAC-M-186, July, 1964.

An extensive bibliography for the entire field of queueing theory is to be found at the end of Saaty's book, reference (18) above.

CS-TR Scanning Project
Document Control Form

Date : 12 / 11 / 95

Report # LCS-TR-21

Each of the following should be identified by a checkmark:
Originating Department:

☐ Artificial Intellegence Laboratory (AI)
☒ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) ☐ Technical Memo (TM)
☐ Other:

Document Information

Number of pages: 108 (115-IMAGES)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☐ Single-sided or
☒ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☒ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☐ Unknown ☐ Other:

Check each if included with document:

☒ DOD Form ☒ Funding Agent Form ☒ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other:

Page Data:

Blank Pages(by page number):

Photographs/Tonal Material (by page number):

Other (note description/page number):

Description : Page Number:

IMAGE MAP: (1-108) UN#'ED TITLE PAGE, 2-45A, UN#'ED BLK, 45B-106
(109-115) SCANCONTROL, COVER, FUNDING AGENT, DOD, TRGT'S (3)

Scanning Agent Signoff:
Date Received: 12/11/95 Date Scanned: 1/8/96 Date Returned: 1/11/96

Scanning Agent Signature: Michael W. Cook

Rev 9/94 DS/LCS Document Control Form cstrform.vsd

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R&D			
(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)			
1. ORIGINATING ACTIVITY (Corporate author) Massachusetts Institute of Technology Project MAC		2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	
		2b. GROUP	
3. REPORT TITLE Queueing Models For File Memory Operation			
4. DESCRIPTIVE NOTES (Type of report and inclusive dates) Master's Thesis, Electrical Engineering			
5. AUTHOR(S) (Last name, first name, initial) Denning, Peter James			
6. REPORT DATE November 1965		7a. TOTAL NO. OF PAGES 108	7b. NO. OF REFS 18
8a. CONTRACT OR GRANT NO. Office of Naval Research, Nonr-4102(01) b. PROJECT NO. Nr-048-189 c. d.		9a. ORIGINATOR'S REPORT NUMBER(S) MAC-TR-22 (THESIS)	
		9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	
10. AVAILABILITY/LIMITATION NOTICES Qualified requesters may obtain copies of this report from DDC.			
11. SUPPLEMENTARY NOTES None		12. SPONSORING MILITARY ACTIVITY Advanced Research Projects Agency 3D-200 Pentagon Washington, D.C. 20301	

13. ABSTRACT

A model for the auxiliary memory function of a segmented, multiprocessor, time-shared computer system is set up. In particular, a drum system is discussed, although no loss of generality is implied by limiting the discussion to drums. Particular attention is given to the queue of requests waiting for drum use. It is shown that a shortest-access-time-first queue discipline is the most efficient, with the access time being defined as the time required for the drum to be positioned. Time is measured from the finish of service of the last request to the beginning of the data transfer for the present request. A detailed study of the shortest-access-time queue is made, giving the minimum-access-time probability distribution, equations for the number in queue, and equations for the wait in the queue. Simulations on CTSS were used to verify these equations; the results are discussed. Finally, a general Markov Model for Queues is discussed in an Appendix.

14. KEY WORDS

Computer	On-line computer systems	Time-sharing
Machine-aided cognition	Queueing models	Time-shared computer systems
Multiple-access computers	Real-time computer systems	

DD FORM 1 JAN 64 1473 (M.I.T.)

UNCLASSIFIED
Security Classification

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**